Tributary: Towards Interactive Real-Time Analytics of Large Scale Sensor Time Series Data

by

Yadid Ayzenberg

B.Sc. Mathematics and Computer Science
Ben-Gurion University, 1999

Master of Business Administration
Tel-Aviv University and Northwestern University, 2008

M.Sc. Master of Science
Massachusetts Institute of Technology, 2012

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning.
in partial fulfillment of the requirement for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2016

Author Signature redacted
Program in Media Arts and Sciences
Nov 24, 2015

Certified by Signature redacted
Rosalind W. Picard
Professor of Media Arts and Sciences
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by Signature redacted
Pattie Maes
Associate Academic Head
Program in Media Arts and Sciences

Tributary: Towards Interactive Real-Time Analytics of Large Scale Sensor Time Series Data

by
Yadid Ayzenberg

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning, on November 24, 2015
in partial fulfillment of the requirement for the degree of

Doctor of Philosophy

Abstract

State of the art technology has made it possible to monitor various physiological signals for prolonged periods. Using wearable sensors, individuals can be monitored; sensor data can be collected and stored in digital format, transmitted to remote locations, and analyzed at later times. This technology may open the door to a multitude of exciting and innovative applications.

We could learn the effects of the environment and of our day-to-day choices on our physiology. Does the number of hours we sleep affect our mood during the following day? Is our performance impacted by the times we schedule our recreational activities? Does physical activity affect our quality of sleep? Do these choices have an impact on chronic conditions?

This proliferation of smart phones and wearable sensors is creating very large data sets that may contain useful information. Gartner claims that the Internet of Things Install Base Will Grow to 26 Billion Units By 2020. However, the magnitude of generated data creates new challenges as well. Processing and analyzing these large data sets in an efficient manner requires advanced computational tools. The challenge is that as more data are collected, it becomes more computationally expensive to process requiring novel algorithmic techniques and parallel architectures. Traditional analysis techniques do not scale adequately and in many cases researchers are required to create customized environments.

This thesis explores and extends the affordances of warehouse scale computing for interactivity and pliability of large-scale time series data sets. In the first part of the thesis, I describe a theoretical framework for distributed processing of time-series data that is implementation invariant and may be implemented on an existing distributed computation infrastructure. Next, I present a detailed architecture and implementation of the theoretical framework, which was deployed on several clusters, as well as in-depth analysis of the user-interface design considerations and the user experience design process.

In the second part of the thesis, I present a system evaluation that consists of two parts. The first part is a quantitative characterization of the system performance in a variety of scenarios that included different dataset and cluster sizes. The second part contains the results of a qualitative user study: researchers were asked to use the system to analyze data that they had collected in their own studies and to participate in an ethnographic study on their experience.

This study reveals that distributed computing holds great potential for accelerating scientific research utilizing large scale sensor data sets, providing new ways to see patterns in large sets of data, and much speedier analyses.

Thesis Supervisor: Rosalind W. Picard
Title: Professor of Media Arts and Sciences, Program in Media Arts and Sciences, MIT

Tributary: Towards Interactive Real-Time Analytics of Large Scale Sensor Time Series Data

by

Yadid Ayzenberg

Reader____ Signature redacted

Andrew B. Lippman

Senior Research Scientist

MIT Program in Media Arts and Sciences

Reader_____ Signature redacted

John Roese

Senior Vice President and Chief Technology Office

EMC

Acknowledgements

No thesis would be complete without acknowledgements and this thesis is no exception. This thesis is the fruit of several years of research that would have been difficult to carry out in any place other than the MIT Media Lab. The interdisciplinary nature of the work is a result of collaborations across several fields: computer science, bioengineering, data science, and psychophysiology. I have many people to thank; first and foremost, I would like to thank my extraordinary advisor, Rosalind Picard, who advised me and provided me with the opportunity to conduct this research. Having been in industry for many years with little experience in academic research, Roz was my guide to the academic world. She was always pushing me to explore areas that I was passionate about while maintaining academic rigor and being the best that I could be. One of my original research ideas was developing a long term wearable 1-lead ECG in the form of a sleek armband. Although I had very little experience in hardware design, Roz was kind enough to let me explore the space, supported me while I was learning analog electronics, and provided financial resources to build prototypes. Eventually, I abandoned this research direction as I felt that I could leverage my expertise to make a substantial contribution in the field of sensor data analytics. Even though this was a high-risk project that would require significant time and resources, Roz believed that it was an important problem to work on and supported me throughout the process. She always asked hard questions, but never once questioned my ability to follow through and deliver and for that I truly thank her.

I would also like to thank my committee members Andrew Lippman and John Roese. Andy, who was also on my master's committee, has been a wonderful mentor and has always asked difficult questions, forcing me to think very deeply about my research goals. I really value John's technical insights and his ability to quickly assess the broader scope and implications of my research. I'm extremely grateful for his willingness to allocate time from his extremely busy schedule and guide me in my work as well as to connect me to others with overlapping interests.

My study collaborators, specifically Sara Taylor and Brian Mayton, both from the Media Lab, who went out of their way to help me iron out of all the kinks in the first few releases and provide valuable input on features.

I am very lucky to have such great friends in my research group. My dear friend and office mate Karthik Dinakar who always provided moral support when I needed it most as well

as sound technical advise and the occasional Indian food discussion. Also, I would like to thank Javier Hernandez, Akane Sano, and Micah Ekhart for their friendship and helping me get through the rough patches during my PhD research.

I am grateful to Frank Moss, who believed in me and admitted me to the Media Lab as a student in his New Media Medicine research group and gave me the unique opportunity to do groundbreaking research and to John Moore my fellow group member who provided friendship and support during my first year in the lab.

I would like to acknowledge Patrick Henry Winston. I took Patrick's course, the human intelligence enterprise, and subsequently was his teaching assistant a year later. Patrick taught me much more than classic AI theory. He taught me how to communicate my ideas clearly and concisely to a wide audience and that is a truly valuable skill. He was also kind enough to review my slides before important presentations and provide insightful comments.

My close friends from the lab throughout the years who added much joy to this journey: Nadav Aharony, Eyal Shachar, Julia Ma, Amir Lazarovich, Roy Shilkrot, Guy Satat, Tomer Weller, Mark Feldmier, and Oren Lederman. I would also like to thank my wonderful friends from all over MIT and Cambridge: Max and Maya Little, Esti and Ami Yaacobi, Liora and Erez Shmueli, Giora and Lia Alexendron, Yuval and Irit Tal, Michal Lazarovich, and Esteban and Adriana Castro Izquierdo. Also, many thanks to Linda Peterson and Keira Horowitz from MAS who made sure I had everything I needed to successfully graduate.

I would like to thank my amazing parents and siblings: my parents Oded and Aviva Ayzenberg for providing me with an environment to flourish, teaching me to be curious and giving me all the love in the world and my brother and sister: Tomer and Maya for being my best friends. Also, my wife's parents Ami and Sara Shoshan for supporting us during our transition to the US and helping us to get settled in.

Finally, I could not have done any of this, nor would it be worth doing without my family: my daughters Noga and Ori who provided me with unconditional love and put everything else in perspective and my amazing wife Ifaat, for whom I'm forever grateful for supporting me and enabling me to pursue my dream of coming to MIT, and bearing the hardships of being in a foreign land away from family and friends. She managed to maintain a demanding career, be an incredible mother, and supported me so I could work on my research. Ifaat, your love and friendship enabled me to flourish, this thesis is dedicated to you – I love you.

Table of Contents

List of Tables

List of Figures

There are two possible outcomes: if the result confirms the hypothesis, then you've made a measurement. If the result is contrary to the hypothesis, then you've made a discovery.

Enrico Fermi

Introduction

State of the art technology has made it possible to monitor various physiological signals for prolonged periods. Using wearable sensors (Sazonov & Neuman, 2014), individuals can be monitored; sensor data can be collected and stored in digital format, transmitted to remote locations, and analyzed at later times. This technology may open the door to a multitude of exciting and innovative applications. We could learn the effects of the environment and our day-to-day actions, and choices on our physiology. "Does the number of hours we sleep affect our mood during the following day?" "Is our performance impacted by the times we schedule our recreational activities?" "Does physical activity affect our quality of sleep?" "Do these choices have an impact on chronic conditions?"

This proliferation of smart phones (Fitchard, 2013) and wearable sensors is creating very large data sets that may contain useful information. Gartner claims that the Internet of Things Install Base Will Grow to 26 Billion Units By 2020 (Middleton, Kjeldsen, & Tully, 2013). However, the magnitude of generated data creates new challenges as well. Processing and analyzing these large data sets in an efficient manner requires computational tools. Signal processing is often used to analyze this data. This may include: Smoothing for noise artifact removal, peak detection, filtering and others. The challenge is that as more data are collected, it becomes more computationally expensive to process requiring either novel algorithmic techniques or utilizing parallel architectures.

Two years ago, the Framingham Heart Study (Dawber, Meadors, & Moore, 1951) lost $4 million (a full 40 percent of its funding) from the federal government due to automatic spending cuts. This seminal study, begun in 1948, set out to identify the contributing factors to

Cardiovascular Disease (CVD) by following a group of 5,209 men and woman and tracking their life style habits, performing regular physical examinations and lab tests. This study was responsible for finding the major risk factors for CVD, such as high blood pressure and lack of exercise. The costs associated with such large-scale clinical studies are prohibitive (Collier, 2009), making them accessible only to organizations with sufficient financial resources or through government funding. One of the major cost drivers in these types of studies is data acquisition and management.

These high costs, which inhibit the collection of data, create the chicken and egg problem of clinical data collection: if you do not know whether a piece of data is clinically relevant, you do not collect it. But if you do not collect it, it will be difficult to determine whether it is clinically relevant. Cost and complexity prevent us from gathering all of the data, so we gather only data that is widely known to be relevant, which limits its usefulness in discovering new types of correlations. This makes it very challenging to determine which data are pertinent to a specific clinical state on top of what is already scientifically proven. On the other hand, the collection of additional data would increase its statistical power and enable the discovery of new correlations.

Historically, clinical scientific discovery was mostly done in small and incremental steps (Lederman, 2014) (Covinsky, 2013) : a hypothesis was formed, data was collected from a group, and the hypothesis was proved or nullified. This resulted in small data sets that contained limited statistical power. What further added to this problem was data "silofication". Relevant data may have been collected but would be stored in separate systems that would not enable easy access for analytical purposes (Bresnick, 2014). Take the case of a hospital, for example: patient history would be stored in one system, imaging in another, and prescription refills in yet another. This barrier was mostly generated due to technological discrepancies. The nature of the data dictated the type of database that would be used to store it.

However, technological advances in sensor technology and the pervasiveness of mobile phones decrease the costs of data collection, making it possible to collect large data sets from the population and perform exploratory analysis. Modern data storage architectures can store different types of data in a single database. What's more, the current, widely available big data

computation infrastructures are enabling the analytics of enormous quantities of data in ways that were never possible before. By gathering and analyzing individual data and comparing it to data of a population, we may be able to classify disease occurrences and predict health outcomes.

An example of a study that utilizes some of these technologies is the UCSF Health eHeart (Leland, 2013). This study sets out to develop new and more accurate ways to predict heart disease based on measurements, behavior patterns, genetics, and family and medical history. A second goal is to understand the causes of heart disease (including heart attack, stroke, heart failure, atrial fibrillation, and diabetes) and find new ways to prevent it. Some of these goals are similar to the original Framingham study, but in contrast to that study, new technologies are utilized to obtain the data.

But what of interacting with the data itself? Is the ability to store a massive data-set in file server based storage, and accessing it on a file by file basis sufficient for unlocking the value that the data holds? Will using traditional computational methods enable us to fully utilize the data? Are there additional ways that we can interact with a dataset without the need to fully iterate over its entire contents?

This thesis explores and extends the affordances of warehouse scale computing for interactivity and pliability of large-scale time series data sets. In Chapter 1, I discuss the motivation and background for this work, and lay out the challenges associated with analyzing large-scale sensor time-series data sets. The 2nd Chapter contains details of prior work in the field of time series storage and analysis tools, and distributed processing frameworks. In Chapter 3, I describe a theoretical framework for distributed processing of time-series data that is implementation invariant and may be implemented on an existing distributed computation infrastructure. This framework is the foundation for Tributary – a system that enables researchers to store and analyze large-scale sensor data sets. In Chapter 4 I present an in-depth analysis of the user-interface design considerations and the user experience design process. Chapter 5 describes the Tributary detailed architecture and implementation of the theoretical framework that was deployed on several clusters.
In Chapter 6, I present a system evaluation that consists of two parts. The first part is a

quantitative characterization of the system performance in a variety of scenarios that included different dataset and cluster sizes. The second part contains the results of a user study: researchers were asked to use the system to analyze data that they had collected in their own studies and to participate in an ethnographic study on their experience. The thesis concludes with Chapter 7 in which I present the thesis contributions, outlook and future work.

"Engineering, medicine, business, architecture and painting are concerned not with the necessary but with the contingent - not with how things are but with how they might be - in short, with design."

Herbert Simon

1 Background and Motivation

1.1 The 'Crit Day' study

Public speaking is often associated with high levels of stress and anxiety. Many individuals fear standing on the stage, being in the spotlight, and forgetting what they meant to say. Additional elements that may add to the anxiety are the possibility of being asked questions that they are not prepared to answer. We set out to characterize the skin conductance changes related to public speaking. "Would it be possible to find a correlation between speakers' perceived stress levels and their physiological response?" "Were the levels of stress highest before, during or after a talk?" "Were the physiological responses of individuals that perceived themselves as very stressed higher than the ones of those that perceived themselves as calm?"

Each year, the MIT Media Lab organizes an event in which second year master's students present their thesis proposal to all of faculty, students, and researchers. Each proposal is evaluated in terms of depth, originality, and contribution. At the beginning of this seminar, the students are told of the importance of their 'Crit Day' performance that will be a factor that weighs heavily in their Ph.D. application. Naturally, this adds significant levels of stress to the students who realize that this speaking event may determine the future of their academic career. We decided that 'Crit Day' was a valuable opportunity to measure the physiological effects of public speaking.

We recruited 11 graduate students, 8 males and 3 females, who were designated to present on 'Crit Day'. During the study we collected Skin conductance, skin temperature, and 3-axis accelerometer – we used the Affectiva Q^{TM} wristband sensor. The data was recorded at a sampling rate of 8 Hz, and was recorded using dry Ag-AgCl electrodes. Each participant

received a pair of sensors, one for each wrist, so we could collect bi-lateral data. The participants were asked to wear the sensors for 72 hours starting on the morning of the day before their presentation. In addition, we interviewed each participant a few days after the end of the measurements. During those interviews, we asked the participants to describe their experiences during the 72 hours. We asked them to note any unexpected events or events that caused them a great deal of anxiety or emotional strain.

We recorded the exact times of presentation and Q&A sessions after the presentation for each participant. We asked the participants to maintain journals of what had happened to them during the three days of the study. Upon the completion of the study, we performed analysis of the data. For each participant we had collected around 45MB of data and several self-reports. This resulted in a total data set of 495MB of sensor data for all study participants. For each participant we the sensor generated 5 files on overage for the period of the study, resulting in a total of 55 files of sensor data.

Our results suggested that the level of perceived stress was the highest the day before the presentation, and was lowest the day after the presentation. On the day of the presentation, the perceived stress level was between the two. We tried to use the number of peaks as a measure of stress, but there wasn't a correlation between this measure and the participants' perceived stress level.

While this is a relatively small study in terms of number of participants, it generated a fair amount of data due to the longitudinal nature of the recordings. The analysis of this data set required significant effort and time. When we wanted to test a hypothesis, we had to execute our scripts that read the data from files on a disk into RAM. This process repeated itself each time we formulated a new hypothesis. This process was completely stateless. Each iteration was the result of coding a new script and re-executing it on the data. There were cases in which we would alter one or more parameters in order to observe their effect on the final result. We had to save the result at the end of such an iteration in order for us to compare between the iterations – and maintain a record of what parameters were changed to achieve that result.

1.2 FEEL

The motivation for creating FEEL was to see if a system could be created that would capture the users' context in an unobtrusive manner by using a mobile phone and then combine that context with a physiological signal to provide insights into the changes of the signal. The mobile phone is capable of obtaining partial context; It is aware of whether or not a user is having a phone call, if the user is at a certain location, if the user is browsing the web or using an app or reading an email, but it is not aware of other occurrences that are more minute. The resolution of context obtainable by a mobile phone is such that it provides a high level picture of what the user was doing. We may still need the user to label and annotate the signal to achieve a higher resolution. At times the high level picture will provide sufficient information as to whether or not a specific event contributed to the change of the biological signal.

The idea was that by providing users with the biophysiological signal combined with the contextual information, they would be able to better recall which event occurred during that time. Even if we could not record the specific context because the phone lacks the sensors and logic to decode that context, we still have other events before and after that could serve as anchors or memory prosthetics for the user and enable him to gain insight into the data and annotate and label those events.

This idea was tested by setting up a study comparing two groups of users, one that could only access the biophysiological signal and the other that could access both the signal and contextual information acquired by their mobile phone. Both groups were required to label the signal in terms of their valence and arousal as well as rate the confidence of their ratings. If the system combining biophysiology with context works better than the one with biophysiology only, then we would find validation for our idea. Later in this thesis we describe the specific study and its component hypotheses.

The approach was to create a mobile application that runs un-obtrusively on the mobile phone as a service. The application is constantly recording contextual information. It is aware of changes in location, phone calls that the user is holding, calendar entries and meetings that appear on the calendar, and emails that the user is reading. Any one of those events will be

recorded and can later serve as a reference point for annotating the physiological sensor recording. The second element of the system is a web platform that combines the contextual information and physiological information and enables the user to view the contextual information overlaid on top of the physiological data in a user-friendly interface. The user can determine where he was or what he was doing and view the changes that occur in his physiology during those times.

The methodology was to test the system out in the wild, in contrast with testing it in a laboratory setting where everything could be recorded and annotated. The goal was to see if it was possible to record the users in their everyday experiences and whether the system could determine the contextual information and provide the user with a tool that would assist them in annotating the data and also assist in providing short term insights from the data. As part of the system evaluation we ran a user study with 10 participants. Participants were asked to wear a pair of commercial electro-dermal activity (EDA) wrist biosensors on their left and right wrists for a period of 10 days. The sensors measure EDA, skin temperature and 3-axis accelerometer and store the readings on an internal SD card. All of the sensors were time synced using the same machine in order to synchronize between the left and right sensor readings. Participants were instructed to wear the sensors for as long as possible and to take them off and charge them during bathing times.

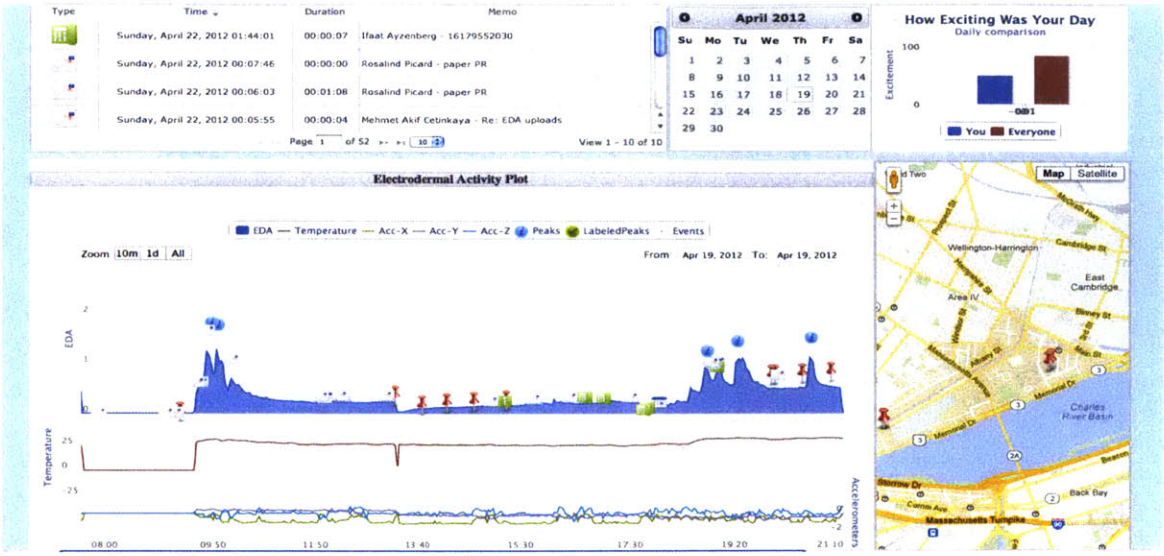


Figure 1. The FEEL web user interface

The resulting dataset size was 1.25GB of physiological sensor data (125MB per person) in addition to several tens of megabytes of contextual data (GPS, calendar, email, and phone logs). The data were stored both as discrete sensor data files, and in a MySQL database. Several hypotheses were tested. One of these was: is there a correlation between a user's self-reported arousal and their arousal as measured by a sensor? But what feature in the signal is indicative of daily arousal level? Is it the maximum amplitude of the signal for that day? To that end, several features were extracted from the signal as a measure of daily arousal:

1. Total area under the curve – using the trapezoidal rule to approximate the total area under the EDA curve for each day
2. Most recent area under the curve - using the trapezoidal rule to approximate the total area under the EDA curve for only the most recent 25% of the measured signal for each day
3. Recent area under the curve - using the trapezoidal rule to approximate the total area under the EDA curve for only the recent 50% of the measured signal for each day
4. Max EDA – finding the maximal amplitude of the EDA signal recorded for each day

For each of these features, a correlation coefficient was calculated for the feature and the self reported daily arousal over the study period of 10 days. During the analysis we came up

with additional hypotheses that the arousal as reported by the user was also correlated to their ability to correctly identify their emotional state. We decided to test this hypothesis by (1) calculating the correlation of each EDA feature and the self reported arousal level, and then (2) testing if result of (1) correlated with the user's Toronto Alexithymia Score (TAS) (Bagby, Parker, & Taylor, 1994), which is a measure of their ability to identify their emotional state. We found that there is a correlation between a person's ability to determine their own arousal level and their score on the Toronto-alexithymia test: the less alexythymic they were, the better their correlation between the EDA and their self-reported arousal.

The process described in the previous section repeated itself in this case as well. With each iteration of our investigation, we had to modify our script, execute it, and record the results for comparison with prior and future iterations. The execution itself took several hours, as we were running the analysis on a single machine. This process required significant manual effort on our part. Could this paradigm be changed? Would it be possible to devise hypotheses and test them with very little time and effort?

1.3 The Sensor Data Processing Pipeline

As part of the preliminary research, I conducted an ethnographic study in order to characterize the sensor data processing pipeline. The study group contained researchers who had managed sensor-based studies. Several had done human participants studies utilizing wearable sensors. I asked each of them to describe the steps they performed, throughout the analytics process. I compiled the results of this survey into the diagram in figure 2.

Format conversion: in most cases the data need to be converted from a sensor vendor proprietary format, which is usually binary, into a format that can be ingested by other tools. This is usually a textual format such as Comma Separated Values (CSV). The goal of this step is to produce data that can be ingested by other analytics tools or databases. In many cases the conversion is initiated manually on a file-by-file basis and is done by using a tool with a graphical user interface provided by the sensor vendor. These tools are designed for small-scale studies with few participants.

Labeling: the meta-data of each sensor stream is recorded. This includes items such as sample-rate, start-time, end-time, and participant-id. Sometimes the meta-data are recorded as part of the file name, or as a header within the file. In other cases the meta-data are recorded in an external excel sheet, or if the data are recorded in a databases, the meta-data will be part of the record.

Time synchronization: in case of multiple sensors for a single participant, the data need to be time synced. In many cases, the Real Time Clock (RTC) in the sensor exhibits some drift due to factors such as power source stability and various environmental effects such as temperature. The process of synchronization includes determining the actual time offset for a particular sensor data stream, and updating each timestamp with this offset.

Merge/Partition: the signals are partitioned into chunks, which are usually on the scale of minutes or hours. Chunks based on specific characteristics such as time or event onset/offset are selected for processing.

Analytics tool ingestion: the chunks are ingested into an analytics tool such as Matlab, Python or R.

Sparse visualization and automatic Inspection: the signals are inspected either visually or automatically by utilizing scripts or other analytics tools. The purpose of this inspection is to determine whether the signal is valid and not corrupt, and to detect outliers.

Outlier discarding: in cases where a stream contains values that are outside the range of expected values, the user may discard the entire stream or a segment containing the abnormal values.

Denoising: in the next stage of the pipeline, the signal is denoised. This is done to remove various artifacts, such as movement (in the case of wearable sensors) from the signal.

Distribution calculation: in some cases, one or more distributions are calculated and the signals are normalized.

Windowing and Feature extraction: features are extracted from the signal usually on the basis of a sliding a window, which is a function of two variables: a discrete time index, and a width that represents a period of time.

Predictive analytics / Trend analysis: analytics are applied in order to perform prediction or to identify past patterns.

Each of the steps above may take anywhere from several minutes to days - depending on the amount of data, the computational complexity of the algorithms, and the computational resources that are at the disposal of the researcher. In many cases, this is a trial-and-error process that is hindered by the length of time that each step takes. What's more, the steps are not easily parallelizable; much time is spent on very heterogeneous steps, which may involve different parts of the data.

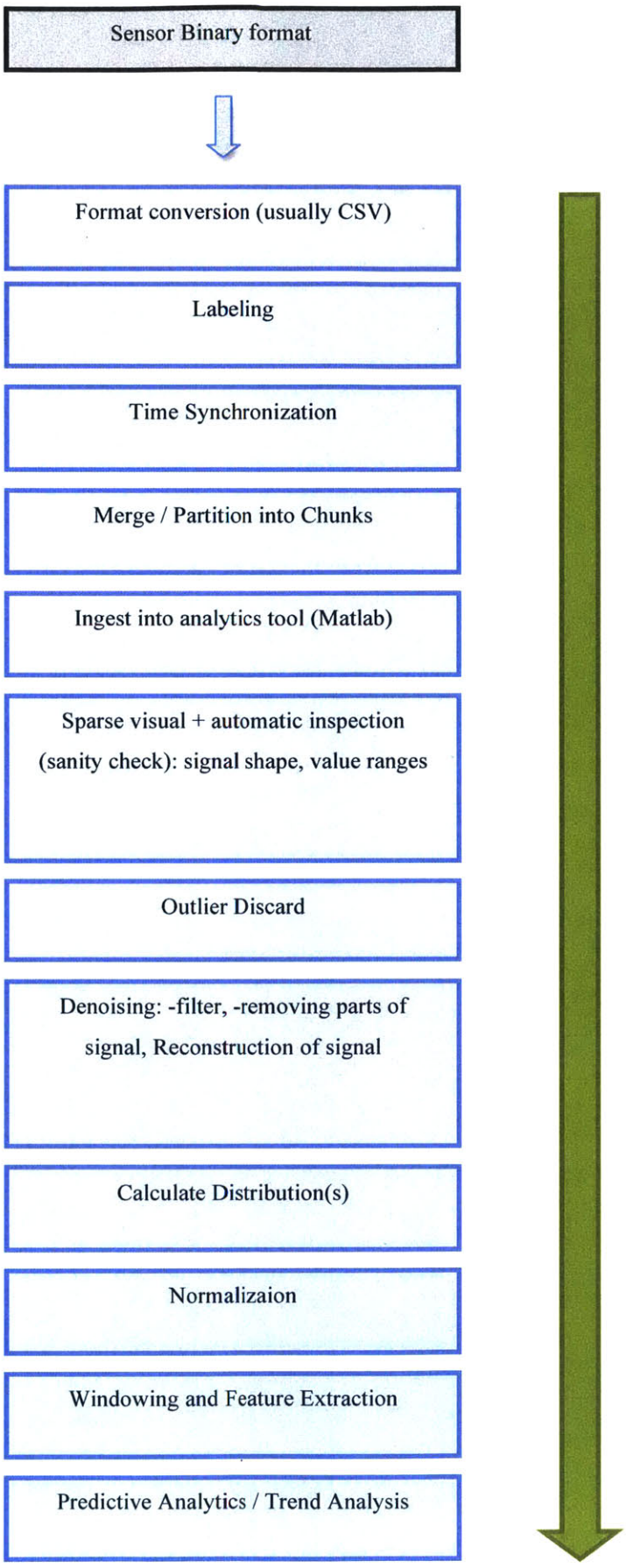


Figure 2. Biophysiological Data Processing Pipeline

In addition to the researchers having to write, test and maintain an analytics pipeline they also have to decide on a mechanism for the storage of the sensor data. Most of the researchers in the study used one of the two storage formats;

1. File-system storage – the researchers decide on a directory structure and naming convention for each file. The files are usually placed either in central location (such as a file server) or shared by some other means such as USB thumb drive or Internet shared (Dropbox, Google Drive) folders.

2. Database system – the researchers decide on a schema and write scripts to import the data into it.

The first mechanism requires very little investment and effort up front as users can create the directory structure manfully, but its downsides are that the data are difficult to query. The second mechanism requires greater initial investment as a database needs to be setup and provisioned, but data are easier to access using a query language (such as SQL).

1.4 Current Approach Limitations

The pipeline in the previous section presents researchers with several challenges:

1.4.1 A Data Management Challenge

1. Multiple formats and systems - the data are stored either on disk as a group of discrete files with varying naming conventions and un-uniform data formats (comma separated value, tab separated value, binary etc), excel sheets, or in a database with a schema that varies from study to study.
2. Data security and access control - setting permissions on the various pieces of data is a daunting task - assuming the researcher is only interested to share part of the data with various individuals.

3. Inventory - how do researchers working on a specific study know where the data are stored and what data are available? How many participant's data are stored, in which date ranges and what sensors? In most cases this inventory needs to be managed manually.
4. Data ingestion - The researcher may need to copy the files from the various sensors and export them to the format of interest. Usually this is a format that can be parsed by the analytics tool.

The above challenges make processing the data a complex task - the researcher must have intimate knowledge of the implementation details of the data-store, and often write large amounts of code to access and clean the data.

1.4.2 A Data Tool-Set and Analytics Challenge

1. In cases where a large data-set is collected, conventional tools are not well suited for exploration and processing of the data. Many environments such as Matlab, R and Julia provide modules for distributed computing, but they require designing code specifically for that environment. The researchers spend a majority of their time writing code and debugging problems in scripts, instead of actually exploring and processing the data (Lohr, 2014). In addition, these environments are ill equipped to handle resilience of very large datasets as cluster nodes may fail in the midst of a computation.
2. In most cases each researcher will design their own scripts to process the data and the reuse of the scripts by others is cumbersome.
3. Most tools do not provide an interface for efficiently interacting with very large data sets.

The Small n Problem - the fixed cost of data analytics

As the costs associated with running large scale experiments may be very high, it is often useful to run an initial pilot study with very few participants in order to prove a hypothesis. Although running a short study with a very small number of participants will result in a relatively small dataset that is computationally inexpensive to processes, it will still require the researcher

to build a set of tools to analyze the data. The cost of this will increase if the data source is new (such as a new type of sensor or new file format) or if the algorithms that are required to process the data are novel. In this case, the cost of developing these tools is fixed and not correlated to the size of the data. This cost may serve as a barrier for performing these types of studies.

The Massive n Problem - the variable cost of data analytics

The number of participants in a study is strongly positively correlated with the size of the resulting dataset. As this number grows, the cost of analyzing the data may become prohibitively expensive as a result of the following reasons:

1. Increased software complexity - in some cases, when the limit of vertical scaling has been reached, extremely large datasets require horizontal scaling. This often entails writing complex distributed software.
2. Hardware / computation expenses - in order to process and store very large datasets, it is possible to either incur a capital expense and procure machines or incur an operational expense and utilize a utility computing provider such as Amazon, Google, Microsoft, or Rackspace.
3. Other operational expenses - such as manpower

1.5 Suggested Approach

1.5.1 Processing and Arduino as a paradigm shift

Processing (Reas & Fry, 2007) is an open source object oriented language and integrated development environment (IDE) built primarily for visual design, electronic arts and new media based art. The project was initiated by Casey Reas and Benjamin Fry in 2001, with the goal of promoting software literacy within the visual arts and visual literacy within technology. Reas and Fry were both students in the Aesthetics + Computation Group at the MIT Media Lab. Processing originated out of their work on Design By Numbers (DBN) (Maeda, 2001), which was created by John Maeda in 1991, who led the group at the time. DBN was a software project aimed at allowing designers, artists and other non-programmers to easily start building computer programs with visual elements.

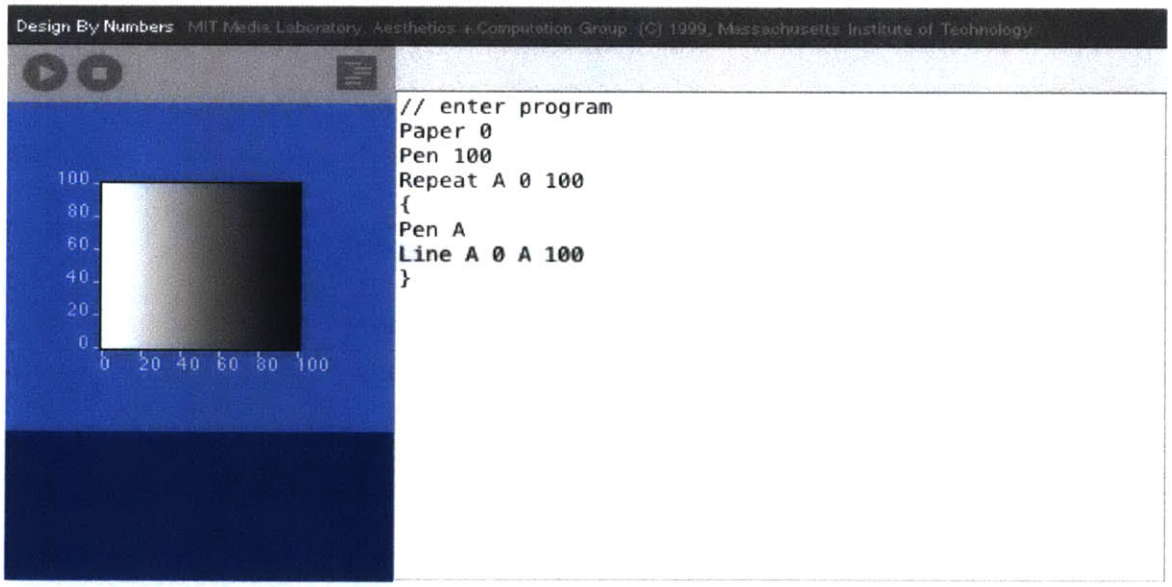


Figure 3. Design By Numbers user interface

Processing is based on the Java programming language and borrows from its flow control statements, variables, and function notations. It provides a single environment which abstracts away many of the complexities associated with programming environments and languages.

Arduino is an open-source integrated development environment and a set of microcontroller based hardware development kits for building devices that can interact with the physical world using sensors and actuators. These kits provide digital and analog Input and Output (I/O) pins that can be interfaced to various expansion boards ("shields") and other circuits. The Arduino IDE is based on the processing project and enables non-programmers to program the hardware boards with a library of easy to use functions. Arduino started in 2005 as a project for students at the Interaction Design Institute Ivrea in Ivrea, Italy. The core team consisted of Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis (Kushner, 2011).

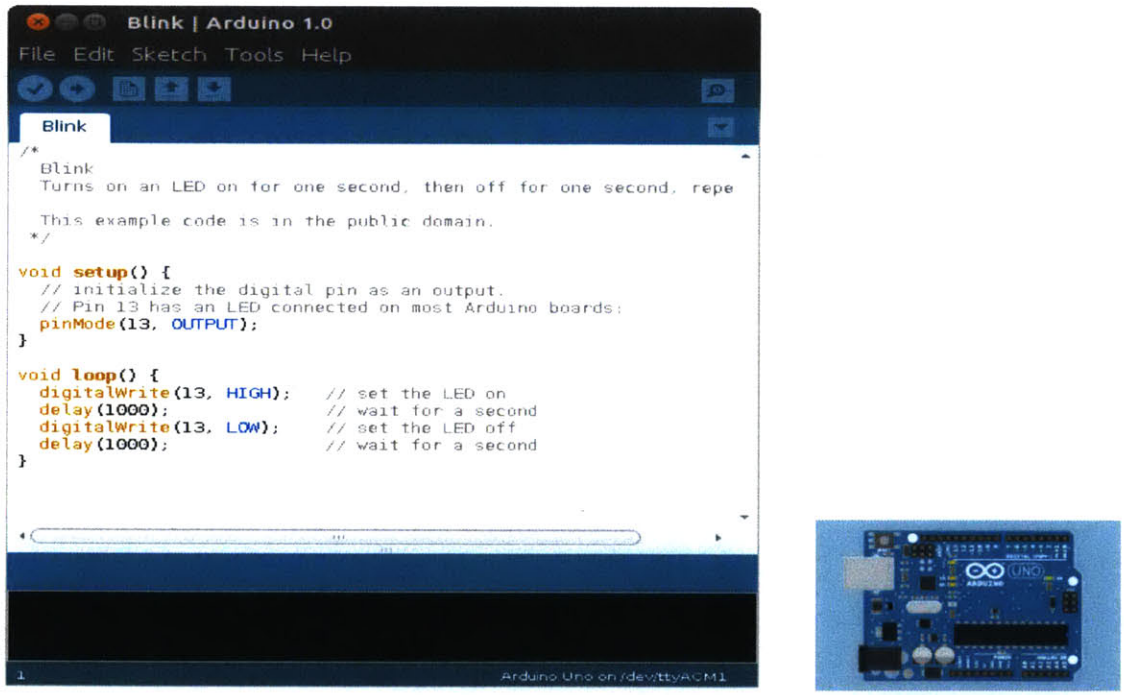


Figure 4. Arduino IDE (left) and Development Board (right)

The main contributions of the Processing and Arduino projects are threefold:

1. **Abstraction** - A library of functions that abstract away much of the complexities required to build visual and physical interactions.

2. **Accessibility**- An integrated development environment that enables non-programmers to install a single tool on their computer, which encompasses the complexities of a programming environment. It is freely available and open-source. The environment contains all of the needed modules to function. In contrast, if users would have wanted to achieve the same functionality without these environments, they would need to download and install a compilation tool-chain (assembler, compiler, and linker), a set of binary libraries from various locations, and a code editor after which they would need to configure various environment variables for the tools to interact with one another.

3. **Portability and Uniformity** – The same environment supports different types of hardware. In both cases the Processing and Arduino environments can run on a variety of OS's (Windows, Linux, Mac OS X) and a variety of computers with different memory, processing and storage capabilities. In the case of Arduino, the same environment can program many types of microcontrollers with different processing capabilities, memory and peripherals, manufactured by different vendors.

One could argue that neither Processing nor Arduino present revolutionary engineering – both are wrappers to a complex environment with a large library for frequently used operations. They are not a basic engineering breakthrough. While this may be the case, these tools are achieving something else. They are revolutionary in another aspect: they are enabling a new way for users to interact with the environment using computers and hardware, even if those users have no or little training as engineers or programmers. They are enabling users to focus on the "what", the end goal rather than the "how", or the method. One could also argue that such tools do not provide enough flexibility, as they abstract away some of the lower levels. However, in many cases – the users are trying to execute some common set of operations that do not require

access to the lower levels of system logic. For instance, a user may be interested in drawing a line on the screen between two coordinates. This can be achieved by providing the user with a line function that may include parameters such as start, end, color and width. In this case, the user has no need to control the individual pixels. In those cases that do need low level control of the hardware or system, thoughtful abstraction can be employed to achieve those goals.

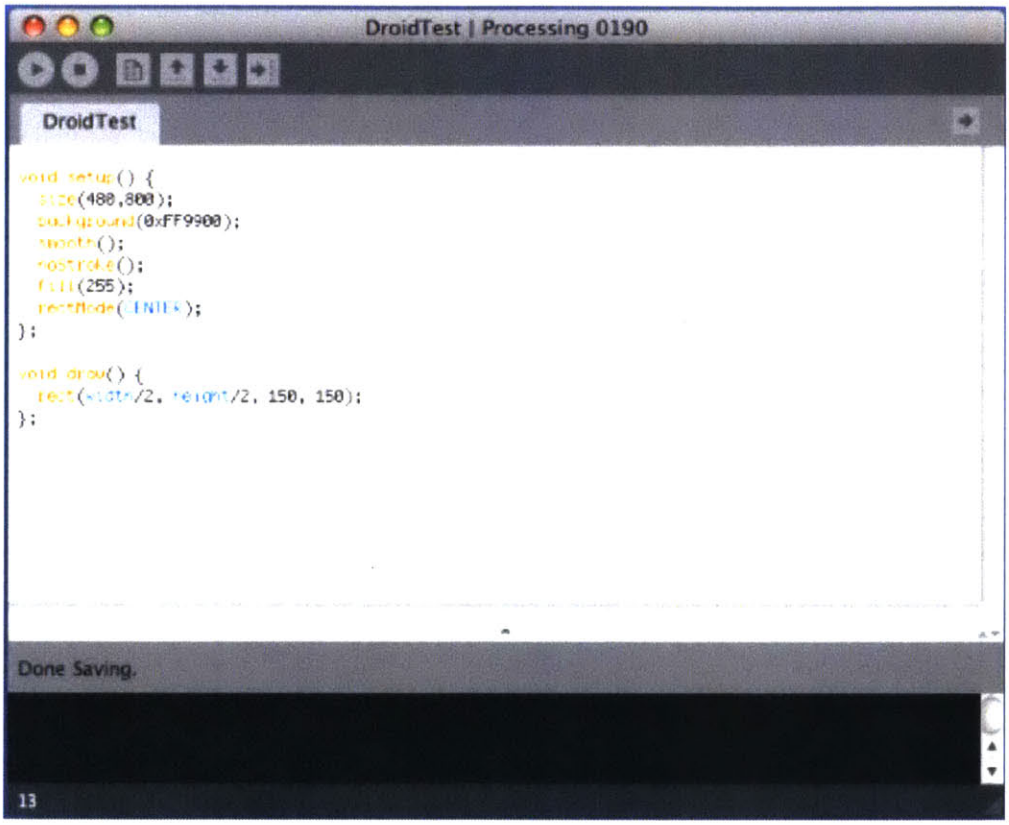


Figure 5. The Processing IDE

Both of these tools abstract the complexity by providing the user with a simple programming model. There are two empty functions that the user is required to implement: setup() and draw() in Processing, and setup() and loop() in Arduino.

The setup() function is executed once upon startup. It will usually contain code to initialize the program. For example setting the frequency of a timer, setting the height and width of the interface window, of setting the initial state of the LEDs on the board are all functionalities that may be implemented in the setup() function.

The draw() and loop() functions are repeated consecutively, allowing the program to change various parameters and react to user inputs throughout its execution. The frequency at which these functions are called can be controlled if need be, but their default setting is adequate in many cases.

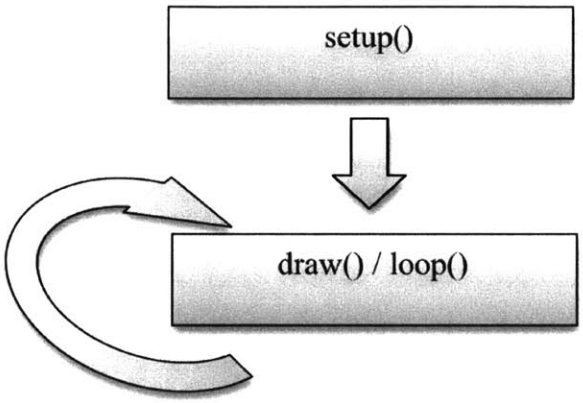


Figure 6. Processing / Arduino program flow

1.5.2 Adoption

The graph (figure 8) (Reas, 2015) below shows the growth in the number of people using Processing on a monthly basis. The dips in the plot are aligned with the start and end of the academic year as Processing is used heavily in academia. The plot shows that in 2006 there were 10,000 Processing users and within a decade that number grew by an order of magnitude to 200,000 in 2015.

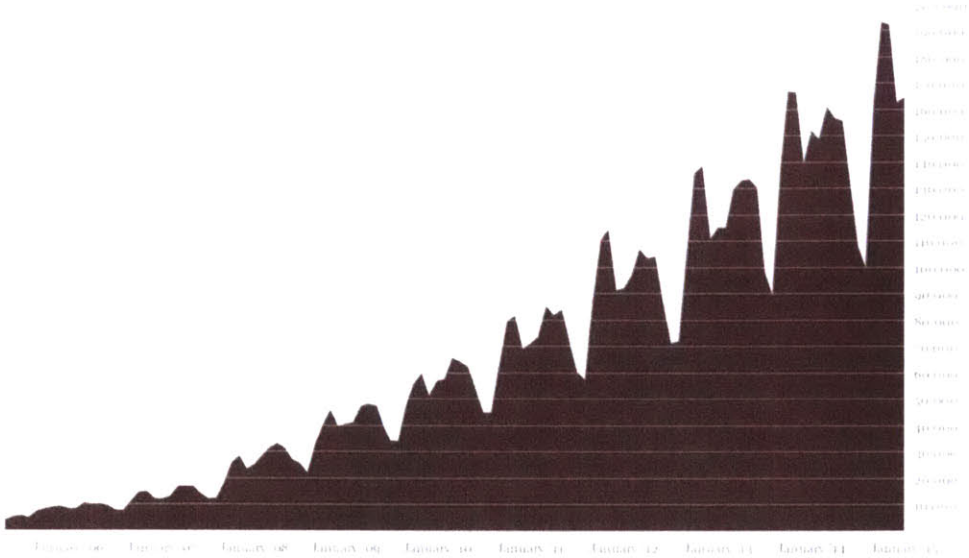


Figure 7. Processing number of monthly users

In 2014, there were 4-5 Million users visiting the Arduino web site over a 3-month period (Orsini, 2014), in addition several millions of boards were sold in 2014, continuing an exponential growth trend as can be seen in the figure below (figure 9) ("Open Source Hardware Summit Speech 2011," 2011).

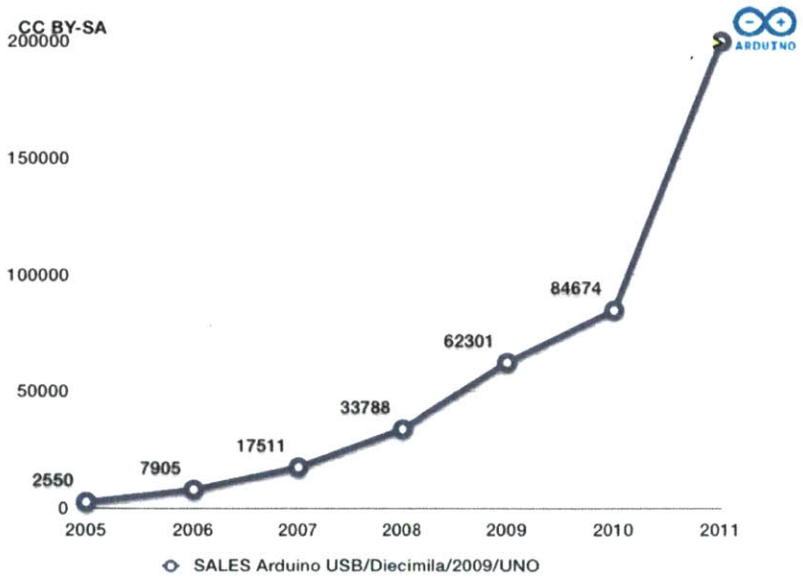


Figure 8. Arduino Sales 2005-20011

Whereas prior to the introduction of Arduino, only professionally trained engineers could build "smart" devices, after its introduction hobbyists, makers and artists could build them as well. While the underlying technology's complexity has remained the same, the abstraction introduced by Arduino has enabled increased adoption by non-traditional users.

1.6 Abstracting the Complexity of Modern Big Data Frameworks

Can the same principles introduced by Processing and Arduino be applied to other fields of technology, such as data science?

Tackling the data deluge created by humans and machines requires significant computational resources. Companies such as Google, Facebook, Twitter and LinkedIn are collecting large amount of data and utilizing advanced Big Data architectures that enable them to store, process, and analyze massive data sets. These companies have also released some of these tools as open-source projects. As a result users can now utilize harness these technologies to analyze their own data sets. Researchers can install these tools in the cloud or within an on-campus data-center. In practice, this is a challenging task; these Big Data frameworks are not easily installed. They require knowledge in networking, OS internals, databases, and Java Virtual Machine (JVM) configuration, to name a few. The also require a steep learning curve – they have unique APIs, programming paradigms, design patterns, and numerous configuration options. Setting up a computing cluster and framework may take several weeks of effort.

In their paper from 2013, Hall et al. present four big challenges that need to be addressed with new abstractions (Hall et al., 2013):

Reliable Structure – representation of core structure so that it can be easily analyzed and visualized at multiple scales, by efficient identification and extraction from complex data.

System Abstractions – language and abstraction design for big data systems that expose a simple programming model which enables adequate performance for the majority of programmers, while allowing access to lower-level details when maximum performance is necessary.

Algorithmic Models – computational abstractions that closely translate to emerging dominating costs (e.g., communication, power, resiliency, precision, heterogeneity).

Uncertainty Management – develop frameworks for efficient management, processing, representation and visualization of big data corpuses that are able to asses their inherent uncertainty and confidence and present the data up to the resolution at which it is reliable.

Figure 9. Rethinking abstractions for big data: why, where. how and what (Hall et al. 2013)

An additional concern comes to mind. Can different fields and different types of users rely on the same abstraction model? Do geneticists and neurologists require the same level of access to the data? For example, a geneticist may be predominately concerned with transactions at the DNA level, while neurologists may be concerned at transactions at the neuron level. If the abstraction level is too low and very similar to the original implementation, users will receive little benefit from the abstraction. Instead, it is useful to find the highest common denominator that will cater to a large variety of users.

It is useful to divide the Big Data abstractions into two classes:

1. Inventory - inventory abstractions are concerned with storage, retrieval and query of the data. They enable the user to access the data at various resolutions and filter it according to various criteria. They enable the users to explore the data and select various parts of the data for subsequent processing. An example of an inventory abstraction is: *retrieve all the data streams of user-id X.*

2. Processing – processing abstractions are concerned with manipulating the data and executing computational operators on the data. They enable the users to perform analytics

on the data. An example of a processing abstraction is: *calculate the correlation between humidity and temperature across all sensor locations.*

The above segmentation is by no means strict; each of the classes may contain abstractions that could possibly reside in the other, but it serves to provide a high-level guideline as to the characteristics of those abstractions.

1.7 Tributary

We wanted to see if it was possible to create a new and improved way for researchers and data scientists to interact with large datasets of time series data. Would such a new approach accelerate the scientific discovery process? Tributary is an attempt to tackle some of the challenges described in the previous section, and to answer this question.

First we designed a theoretical framework to enable users to think about distributed computation of time series data in an abstract and simplified manner. This framework was directly derived from our discussions with researchers as to what kinds of computation they currently preform and the affordances of distributed computation. Next, we designed a system that implemented the theoretical framework and that would attempt to abstract the complexities associated with some of the modern big data frameworks currently being used. The first problem we were tackling was the data management challenge. A large portion of the researcher's effort was geared towards managing the data. Tributary was designed to store the data in a uniform format seamlessly, without requiring the user to decide on schemas, storage media and security mechanisms while enabling the users to easily explore the data and search for specific data segments in an easy way. The second problem we were tackling was the data analytics challenge. In many cases, researchers were not utilizing the cutting edge big data frameworks, as they required a steep learning curve as well as the expertise required to install, provision and maintain a cluster of machines. In addition, users code programming environments are not portable, as they require specific packages that were installed by the user to run. Tributary attempts to

provide the users with a tool that enables them to write operators in a programming language they are already familiar with, while requiring learning only a few basic concepts. As it is provided as a Software as a Service (SaaS) web based platform, the users are not required to install any software on their machines, or on servers. After uploading their data to the system, the users can explore, analyze, or share the data as well as their analytics operators. The environment is shared, and therefore, other users on the platform can execute the users code without needing additional package installations.

The machine does not isolate man from the great problems of nature but plunges him more deeply into them.

Antoine de Saint-Exupery

2 Prior Work

This research lies at the intersection of two fields: time-series storage & analysis tools, and distributed processing frameworks. Existing approaches include a variety of commercial tools and open-source software. This chapter includes a literature review along with a description of the limitations of each approach in the context of interactive analytics of large-scale time-series data sets.

2.1 Time-series Storage and Analysis Tools

Due to the proliferation of sensors and the clinical importance of physiological time-series data, significant work has been done in the field of time series data processing. Iverson introduced APL (Iverson, 1962) a language for processing multi-dimensional arrays that lends itself well to parallel applications. APL uses a non-standard character set requiring customized mappings for support on standard keyboards. The use of APL was eroded partly by the lack of migration paths from high performance mainframes to personal computers as well as the growth of platforms such as MATLAB, Octave, R, and SciLab that have more conventional programming languages. K, a modern derivative of APL, along with kdb (Whitney & Shasha, 2001), a database built on k are used in the financial industry to store and analyze financial time series data (such as stock prices).

Yamamoto and Nakano (Yamamoto & Nakano, 2001) proposed TISAS (Time Series Analysis Supporting system), a system for distributed computation of time-series data. In their system, the user needs to specify on which machine in the cluster the computation will be executed. The user can choose to "export" different datasets to different machines and execute the computation in parallel, but the distribution itself is a manual process. TISAS is based on PVM (Sunderam, 1990) a library for distributed computation in C and Fortran.

Ciccarese and Larizza (Ciccarese & Larizza, 2006) proposed Tempo a framework for the definition, generation and execution of data processing components. Tempo provides data abstractions as well as data processing algorithms that are provided as reusable blocks.

OpenTSDB ("OpenTSDB - A Distributed, Scalable Monitoring System," n.d.) is a distributed time series database built on top of Apache HBase, which is an open-source, distributed, versioned, column-oriented store. It's used for storing and retrieving time series base metrics and has several tools for charting. KairosDB (previously named OpenTSDB2) ("KairosDB - Fast scalable time series database," n.d.) is a time-series database, which was written to work with Cassandra (Lakshman & Malik, 2010), an open-source distributed wide-column store. Both OpenTSDB and KairosDB provide specific aggregators, but other than that they are very limited for performing other signal processing and analytic functions on the time-series data.

McKenna et al. (McKenna, Bawa, Kumar, & Reifman, 2007) proposed PAS (Physiology Analysis System). PAS support advance mining of physiological data and provides a library of data analysis functions that can be executed in a chain as well as user-defined functions. PAS has a monolithic architecture - both the data and the algorithms reside on a single machine, thus limiting the scale of the dataset.

TempoDB ("TempoIQ," n.d.) is a commercial cloud-based platform that enables storage, retrieval and visualization of time-series data. Currently the platform only supports very basic aggregation functions.

Traditional database management systems suffer from many limitations that are applicable to scientific application which require accessing and analyzing large amounts of data. SciDB (Stonebraker et al., 2009) is an array database in which arrays are divided into overlapping partitions. It is meant to tackle some of those limitations by introducing a declarative Array Query Language (AQL) and an Array Functional Language (AFL). AQL supports creation and querying of large array, while AFL supports array manipulation by means of functional operators such as SLICE, SUBSAMPLE, and FILTER.

Zhang et al. (Zhang, Kersten, Ivanova, & Nes, 2011) proposed SciQL, an SQL based query language specifically designed for scientific applications. One of the important innovations is an extension to the value based grouping of SQL:2003 with structural grouping, fixed-size and unrestricted groups based on explicit relationships between elements positions. This addition enables window-based query processing which is applicable to many fields of scientific research.

The main limitation in array database is their proprietary programming language. Some functions are not supported and analyzing the data requires the user to write all of the code using that language. As a result, the code is not portable and difficult to share with other researchers. In addition, it is difficult to utilize other researchers algorithms that were implemented in other more widespread programming languages.

2.2 Distributed Processing Frameworks

An alternative to using a dedicated time-series data processing platform is to utilize a generic distributed processing platform.

Terracotta ("Terracotta | In-Memory Data Management for the Enterprise," n.d.), Gridgain ("GridGain = In-Memory Computing," n.d.) and Hazlecast ("Hazelcast | In-Memory Data Grid," n.d.) are all commercial platforms for large scale cluster in-memory computing. They provide scheduling, message passing and fault tolerance. In theory these could be used for distributed processing of time-series data, however, they provide a very low level API to perform distributed execution. There is no notion of a dataset, but rather a notion of shared memory and computing resources available to perform tasks.

Storm ("Storm, distributed and fault-tolerant realtime computation," n.d.) is an open-source distributed real-time computation system. Similar to the solutions above, it provides facilities for scheduling, message passing and fault tolerance but requires the user to implement data set functionality in the user application level.

Hadoop ("Apache™ Hadoop - reliable, scalable, distributed computing," n.d.) is an open source implementation of Google's MapReduce (Dean & Ghemawat, 2004) programming model that enables batch processing of large datasets. It can be used for time series storage and analysis such as in the systems proposed by dos Santos et el. (dos Santos et al., 2012) and Berard & Hebrail (Berard & Hebrail, 2013). Hadoop is specifically tailored at running programs that utilize the MapReduce paradigm - namely mapping values to keys and then preforming an aggregation of similar keys. This makes it cumbersome for executing generic computational tasks. An additional shortcoming is that Hadoop does not support in memory caching of data sets between several iterations. Each iteration is considered a new job, so the previous stage must be reread from disk, which adds significant latency to the entire process.

Apache Spark (Zaharia, Chowdhury, Franklin, Shenker, & Stoica, 2010) is an open source cluster computing system that offers a general execution model that can optimize arbitrary operator graphs, and supports in-memory computing. Its main advantages over Hadoop are that it support a generic programming model not limited to MapReduce jobs and that it supports in-memory computation and caching. Further more, Spark introduces the notion of Resilient Distributed Datasets (RDDs) which provide fault tolerance at the data set level. If a node computing part of the dataset fails, an alternative node will re-execute that specific portion of the computation. Stratosphere (Ewen, Schelter, Tzoumas, Warneke, & Markl, 2013) offers similar advantages to those of Apache Spark.

There are three significant challenges when utilizing a generic distributed computation framework for time series analysis. First, in many cases there are no facilities for maintaining the temporality of the data. The user has to implement functionality that will ensure the time-series samples maintain their order. Second, the user is required to learn a new programming paradigm (that could take a significant amount of time) such as map-reduce and implement all of the algorithms using that paradigm. Not all algorithms can be implemented using these paradigms. Third, in many of these frameworks, the data are stored in a separate system. The user needs to implement a schema and design software for reading data from that schema.

After growing wildly for years, the field of computing appears to be reaching its infancy.

John Pierce

3 A Theoretical Framework for Distributed Computation of Time-Series Data

A time-series is a sequence of data points, measured typically at successive points in time which are uniformly spaced (Chatfield, 1984). The first goal of analyzing this type of data is for the sake of understanding past behavior which is useful in a multitude of environments. In health-care for instance, we would like to diagnose a patient based on the trends of their bio-physiological data. The data can be either generated by a series of lab tests, physical examinations, or sensor data. The second goal is the ability to perform predictions based on past data. For example, what is the probability that a specific patient will develop a myocardial infarction in the future based on his past ECG recordings. Many of the traditional analytics tools are not optimized for dealing with large temporal natured datasets. The natural temporal ordering of time series data makes analysis distinct from other common data analysis problems, in which there is no natural ordering of the observations.

3.1 The Need for a Framework

Historically, the processing of large-scale time series data was performed on monolithic enterprise scale computers with multiple cores and significant amounts RAM. The challenges associated with this approach are:

1. High computation cost - Barroso and Hölzle (Barroso & Hölzle, 2009) claim that the same compute power is x12 more expensive when using a high-end computer in comparison with a cluster system composed of many low-end computers. They also measured parallel performance of a cluster and concluded that if the application requires more than two thousand cores, then a cluster composed of 512 low-end servers performs within approximately 5% of one built with 16 high-end servers.

2. Lack of elasticity- in some case there may be large variance in the size of the data sets. A high-end monolithic server will be limited by the maximum number of cores and maximum RAM supported by the motherboard. In case processing of a larger data set is required, this physical limit may impose unrealistic processing times. In contrast, when using a cluster, it is possible to purchase additional machines and add them to the network to support larger data sets. Another possibility is to utilize a commodity computation platform such as Amazon EC2 ("AWS | Amazon Elastic Compute Cloud (EC2) - Scalable Cloud Hosting," n.d.), Google cloud platform ("Cloud Computing & Cloud Hosting Services — Google Cloud Platform," n.d.) or Microsoft Azure ("Azure: Microsoft's Cloud Platform | Cloud Hosting | Cloud Services," n.d.) and obtain additional resources on demand without the need to incur capital costs.

With the prevalence of warehouse scale computing platforms such as those used by Google, Facebook and Yahoo and the advent of virtualization – it is possible to perform analytics of large time-series data sets at a very low cost. These types of computing platforms have brought on the emergence of new software such as Apache Hadoop ("Apache™ Hadoop - reliable, scalable, distributed computing," n.d.), Apache Spark (Zaharia et al., 2010) and Stratosphere (Ewen et al., 2013) that enable running distributed and parallel computation.

These software platforms manage the distribution of data across the cluster, the execution of computation on the data, the scheduling of computation tasks, fault tolerance and resilience. However they are not specifically tailored to perform computation on time series data. As such, they do not ensure that the data are partitioned in an optimal manner, and do not provide facilities for maintaining the temporal nature of the data.

The goal of this chapter is to define a theoretical framework that is implementation invariant and may be implemented on an existing parallel computation framework such as those mentioned above. The framework serves as an abstraction layer above that enables user to focus on the analysis of the data, rather than on the underlying infrastructure and its configuration. We begin by providing strategies for partitioning time series across a cluster and follow by describing the various groups of distributed operations that may form the basis for analytics on

time series data. For the sake of simplicity we assume that a time series is univariate and that multivariate time series can be represented by groups of univariate time series (we allow computations to be joint across multiple time series).

3.2 Strategies for Partitioning Time-Series Data Across a Cluster

Before being able to run distributed computations on a data set, it is necessary to partition the dataset across the cluster. Usually the data are to be loaded into the Random Access Memory (RAM) of each of the individual nodes. RAM provides optimal performance in terms of both latency and bandwidth. In case the dataset is larger than the available total RAM of all nodes in the cluster, it is spilled over to persistent storage on each of the nodes.

This is also known as "shipping" the data to the nodes. The Datasets are usually stored in a database or file system, which may be either distributed or monolithic. A brute force approach would be to have a single machine read from the database, and distribute the resulting dataset across the cluster. This machine can either be the cluster manager (sometimes known as the driver) or any other machine that has access to the cluster interface. Although this approach enables easy management of the data shipping process it is limited by both the CPU and IO bandwidth of the reading machine. An alternative to this approach would be to have the master ship only the queries to each of the nodes in the cluster. After receiving a query, each node in turn would query the database and read the resulting dataset to local memory.

The advantage of this approach is that the shipping process is only bound by the IO capacity of the database server. In cases where the database is composed of several machines in a cluster, and the dataset is partitioned across multiple shards, the IO capacity will increase linearly with the size of the database cluster. Lastly, it is possible to maintain a distributed database in which the database nodes and the computation nodes are one. In this case, each node has low latency access to the part of the dataset that it is required to process. The disadvantages of this approach are the increase in complexity in the management of the database and an increase in latency for database writes. Before writing to the database, the client must ask the master where on which node that part of the data should reside, and only then write the data on the relevant node.

When shipping the data to the cluster there are several approaches to partitioning the data:

A. Load each time series fully on one of the nodes (figure 10).

Advantages:

1. Enables processing of a full time series without any network latency
2. Easy to manage – each time series is loaded into the RAM of a single machine

Disadvantages:

1. Uneven load (pressure) on the cluster - if there is significant variance in the sizes of the time series (i.e. some time series are very large and others are small), or if there are more nodes than time series this may lead to sub optimal performance
2. Difficult to manage the cluster pressure

B. Divide each time series into equal blocks, and segment them on the cluster (figure 12).

Advantages:

1. Even pressure on the cluster.
2. Very easy to manage.

Disadvantages:

1. IO / network pressure when applying operations to a full series and when materializing

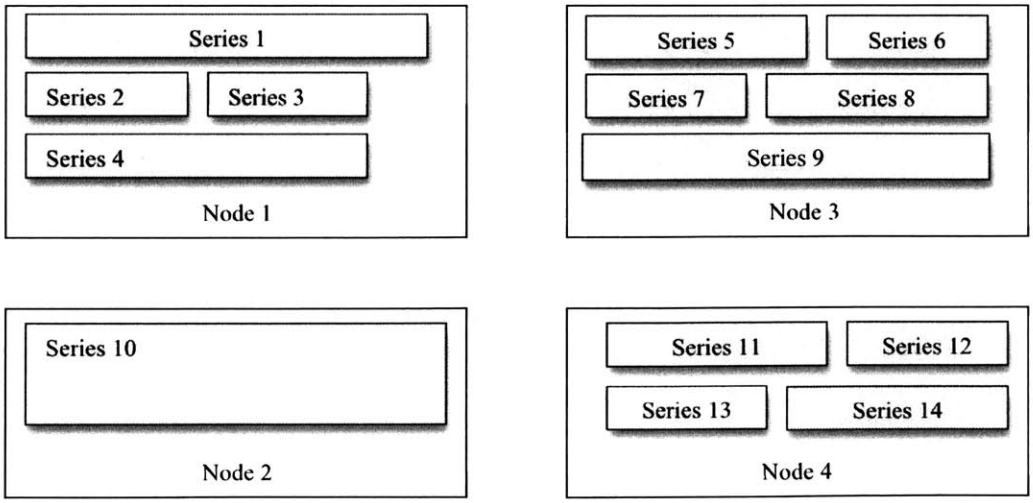


Figure 10. Each time series is loaded in its entirety to a specific node. The cluster is balanced

In the example above, which utilizes the first strategy, it is clear to see that there is even memory and computation pressure on each of the nodes 1 through 4 in the cluster. However consider the scenario below (figure 11):

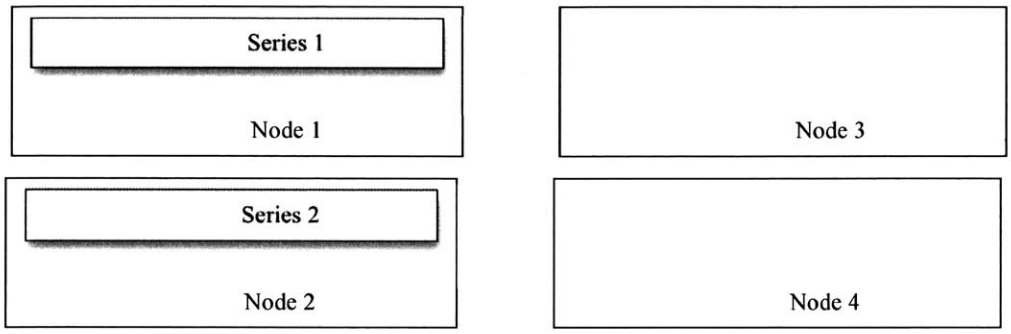


Figure 11. Unbalanced cluster

It is clear from the illustration above that the four nodes are unevenly utilized. Nodes 1 and 2 are heavily utilized while Nodes 3 and 4 are under-utilized.

In the illustration below we utilize the 2nd approach: segmenting each time series into equal sizes blocks, and shipping each block to a different node. Of course, in case there are more blocks than nodes for a specific time series, a node may store several blocks from the same time series. As the illustration demonstrates, each node is evenly utilized.

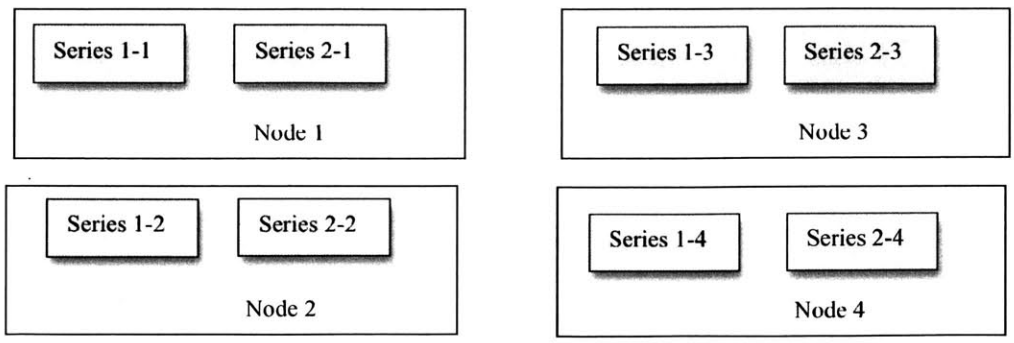


Figure 12. Segmenting each time series results in a balanced cluster

3.3 Executing Computation on Distributed Time-Series

Once the data are distributed across the cluster we can perform computations on the time series. We shall define the following four computation types: Transformations, Aggregations, Combinations, and Selectors.

Univariate Transformations

Given that X is a time series: $X_1, X_2.., X_n$ and X' is a time series: $X'_1, X'_2.., X'_k$

We define a transformation $\int$ as:

$$\int (X) = X'$$

Note that n does not necessarily equal k

We define series S = { S_1, S_2, ... S_n } which is segmented into n blocks, where each block is denoted by S_i.

$$\int (S) = \int (\{S_1, S_2, \ldots, S_n\}) = \{\int (S_1), \int (S_2), \ldots, \int (S_n)\} = X'$$

A transformation of X is the result of applying a function to the time series which would result in X'. For each element in the original X there would be either zero, or one, or more new elements in X'. Transformations are often used to preprocess the data for subsequent analytics. An example of a transformation is an exponential filter, which is often used, for smoothing signals in order to remove noise artifacts. In the figue below (Figure 13) the blue signal is obtained by applying a transformation to the red signal which contains high frequency noise.

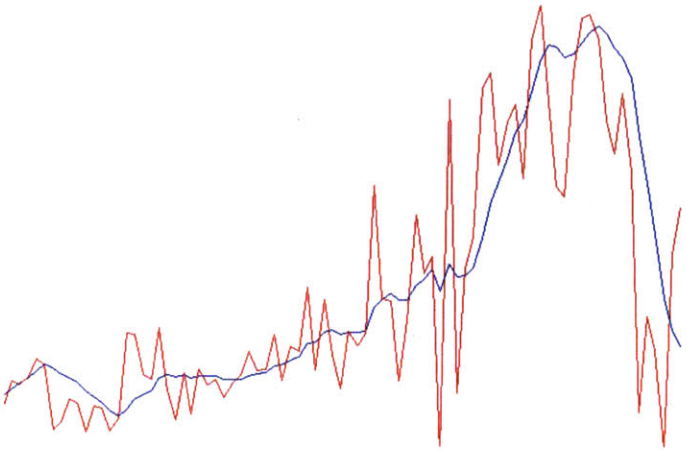


Figure 13. Noisy signal (red) and clean signal(blue)

Applying a transformation to a distributed time series is performed by applying the transformation separately to each block of the time series. There are cases in which the transformations output is based on the input of several past samples or a value that is calculated from a previous iteration of the transformation. In such case, if the size of the blocks is sufficiently large, we could treat each block as an independent time series. In this case, there would be only a minor impact on the output of the transformation. The result of a transformation can be retained in the RAM of each node for subsequent operations, or stored in a database, or file system.

Aggregations

Aggregations are summarizations of the data in a time series. They are used either to gather information about a time series, or to significantly reduce the size of the data. Given that X is a time series of n samples: $x_1, x_2.., x_n$ and A is a list of values: $a_1, a_2, \ldots, a_k$

We define an aggregation $\oplus$ as

$$\oplus(X) = A$$

Note that k is usually significantly smaller than n

Aggregations require two stages of computation: the first, applying an operation to each block of the time series and the second, an aggregation of those results across in the entire cluster. We define series S = { S_1, S_2, ... S_n } which is segmented into n blocks, where each block is denoted by S_i.

$$\oplus(S) = \oplus_s(\{\oplus_b(S_1), \oplus_b(S_2), \ldots, \oplus_b(S_n)\}) = A$$

We shall refer to the first stage as block aggregation ($\oplus_b$) and to the second stage as series aggregation ($\oplus_s$). The series aggregation may either be a mathematical operator (such as summation), or may just store the results of the block aggregations in an array.

In some cases, it is possible to preform the series aggregation with only the results of the block aggregators while in others additional meta-data may be needed. For example consider the aggregator $sum(X) = \sum_{i=0}^{n} x_i$ of some series $x_0, x_2 \ldots, x_n$. In this case the block aggregators perform $sum(B_j) = \sum_{i=j}^{k} x_i$ where B_j is block j of series X. When all the block aggregations have completed, the series aggregators will use the result of each block aggregator and compute $sum(x)$ by adding each of the block aggregation results. In contrast, consider the aggregator $avg(X)$. In this case, we are interested in the average of the entire series. In order to calculate the average correctly, we require not only the results of the block aggregators, but also the number of elements in each block so we can calculate the weighted average correctly.

As the aggregation in executed in parallel, it is important that the aggregator be an associative and a commutative function. An associative function is defined as

$$\int(\int(x, y), z) = \int(x, \int(y, z))$$

and a commutative function is defined as

$$\int (x,y) = \int (y,x)$$

Exhibiting the properties of associative and commutative ensures that the function is computed correctly in parallel, as the final result is an aggregation of aggregations.

It is possible to run non-associative and non-commutative functions across the cluster as well, but this would not utilize the full computation power of the cluster, as the operators would need to be executed in order, and there may be nodes left in idle state waiting for a previous operation to complete.

Examples of aggregations are: sum, mean, multiplication, min, and max. Another example aggregation is area under the curve. Below is an illustration (figure 14) depicting applying an area-under-the-curve aggregation to a time-series.

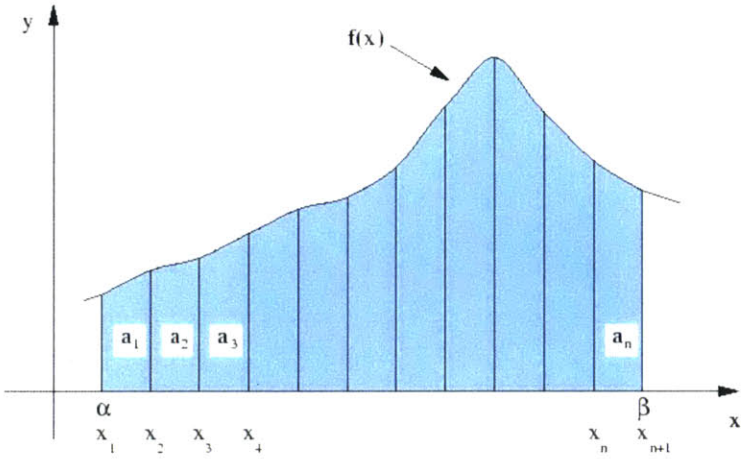


Figure 14. Area under the curve time series aggregation

One method for calculating the area under the curve of a time series f(x) is to calculate the area under the curve of each segment and then sum those areas. The total area equals the sum of the areas of all of the blocks. We can view the time series blocks as segments and sum the areas of each of the blocks across the cluster.

Multivariate Transformations / Compositions

A multivariate transformation (or composition) is an operation that receives as input multiple time series, and outputs either a single time series ($\int_1$) or an aggregation ($\int_2$). More formally: given $X_1, X_2, \ldots, X_n$ where X_i is a time series of n samples: $x_{i1}, x_{i2}.., x_{in}$ and Z is a time series of n samples: $z_1, z_2, \ldots, z_n$ We define the $\int_1$ composition as

$$\int_1 (X_1, X_2, \ldots, X_n) = Z$$

Given $X_1, X_2, \ldots, X_n$ where X_i is a time series: $x_{i1}, x_{i2}.., x_{in}$ and A is a list of values: $a_1, a_2, \ldots, a_k$ We define the $\int_2$ composition as

$$\int_2 (X_1, X_2, \ldots, X_n) = A$$

Compositions are often used when data from several time series are required to process a specific time series. A good example would be to combine an accelerometer time series and an ECG time series recorded from an individual at the same time in order to produce an ECG time series with the movement artifacts filtered out. An example for an aggregation composition is to calculate the correlation coefficient of two time series.

Selectors

A selector is an operation that receives as input a group of time series, and outputs a subset of that group that fulfills a specific predicate. More formally: given $S_1, S_2, \ldots, S_n$ is a group of time series, we define the $\int$ selector as

$$\int (S_1, S_2, \ldots, S_n) = \{S\}$$

Selectors are used to exclude time series that do not match a criteria. For example, if a sensor is producing readings from 0.0 to 1.0 – any value outside of that range would be considered erroneous, and we may consider any time series containing those values as questionable. We may apply a selector to remove any time series containing those values as they may be considered outliers. On the other hand, we may apply a selector to remove any time series that contains only "normal values" as we are interested in analyzing those time series that contain problematic values.

Co-location considerations

Applying a composition on time series in the cluster requires that either matching (identical time range) blocks from different series be collocated on the same node, or that intermediate computation information be exchanged between the nodes, throughout the computation. In order to collocate the matching blocks it is necessary to preform a shuffle, which relocates data from one node to the other. Thus, in case of multivariate transformations, it more efficient to have matching blocks collocated in RAM on the same node initially, rather than preform expensive shuffling operations across the cluster.

3.4 Conclusion

Analysis of time series is an important tool in many fields ranging from medicine, to finance. As the magnitude of data grows due to the proliferation of sensors and increase in storage capacity, the analysis of this data requires novel tools.

In this chapter I have presented a theoretical framework for parallel and distributed processing of large-scale time series data sets. The concepts are implementation invariant and enable the utilization of this framework on a variety of distributed computing environments such as Apache Hadoop and Apache Spark.

If a picture is worth 1000 words, a prototype is worth 1000 meetings.

Tom & David Kelley

4 User Experience Design

4.1 User Personas

"a persona is a description of a fictitious user. A user who does not exist as a specific person but who is described in a way so that the reader can recognize the description and believes that the user could exist in reality. A persona is described based on relevant information from potential and real users and thus pieced together from knowledge about real people interactions" (Nielsen, 2012).

Creating personas as part of user experience design is an important step when designing a UI, which will fulfill the needs of the system's users. The personas descriptions help designer to identify with the users and their unique points of view as well as provide information on who those users are. As part of the design process the following 3 personas were created:

Anne – Anne is a research scientist at a large university. She completed her PhD in bioengineering and has been working at the university for the last 5 years. She studies human psychophysiology by utilizing data obtained from wearable sensors. She is proficient in Python and Matlab and she utilizes them daily for her work. She also utilizes the Internet, mostly for access to research papers as well as for programming reference. Anne has a bit of experience with databases, but no experience with distributed or parallel computing. She has heard about cloud computing, but has never used it herself. Anne usually works with other researchers as part of a study team lead by a PI.

Anne has just completed a long-term human subject study. She has collected files from 1000 participants for 3 months. For each participant, she has collected accelerometer, Electrodermal activity, and GPS data. The files are stored on a local drive and she wishes to start

analyzing the collected data. What she normally does is randomly select files from each participant and visually inspect them. Next, she would execute some scripts on the data trying to determine its quality. Then she would remove artifacts, synchronize timestamps and normalize the data. Anne has formulated several hypotheses and the goal of her analysis is to find evidence in the data to support them. She estimates that even if she is not able to prove the original hypothesis, the data will most likely contain other insights that she is interested in exploring.

John - John is a primary investigator of a lab in a medical research institute. He has an MD and also has a PhD in neuroscience. John does not have a Computer Science background. He has very little programming experience and has written Matlab scripts during his studies. His team is composed of several researchers, experts in bio-physiological signal processing, and several clinicians. John's expertise is mostly in the clinical domain and he is an active physician. He is a resident physician in a teaching hospital.

John is currently managing several large-scale clinical studies. As the studies progress, he is interested in seeing what data has been collected, and is also interested in obtaining a rough estimation of the data quality. He would also like to run some quick analysis to validate whether there is initial evidence that supports the study hypothesis.

Chris –Chris is an undergraduate student majoring in EE/CS in one of the Ivy League universities. As past of his degree, he also works as an undergrad researcher in one of the university's labs that researches future urban design technologies. He is supervised by a graduate student researcher. Chris has some Digital Signal Processing (DSP) experience, is proficient in Java and R, has taken a course on database systems and can write simple SQL queries.

He is currently part of the project team running a study trying to assess urban traffic patterns by utilizing sensors. The project has access to 1 year of data from 5000 sensors across an urban landscape. His role in the project is to obtain the data from the various sensors, validate the

quality of the data and store the data in a relational database that was set up specifically for the project.

The persona Anne represents a researcher with deep technical expertise, who is very comfortable around various programming languages and requires access to study data on all levels. They are responsible for both collecting the data and analyzing it. John represents someone with considerable clinical research skills who has very little experience in computational data analysis. He requires data access at a very high level. Mostly being able to view some high level metrics and aggregations, and perhaps execute a ready-made operator. Finally, the persona Chris represents is a someone with good technical skills who mainly is required to access the data at a low level: data validation and storage.

4.2 Information Architecture

In order to support the various personas a suitable information architecture was designed. While Anne and Chris will most likely be interested in exploring specific streams, or browsing a specific study on a given day, John will most likely be interested in viewing data at a much higher level and viewing multiple studies. Thus, a hierarchical approach was taken: the user may explore each hierarchy independently and browse the elements in that hierarchy. The hierarchies are browsable, and searchable. I refer to browsability as enabling the user to list (or browse through) all the elements in a hierarchy. For example, a user may not know which participant Ids are assigned to a group. As a result they would browse through all the participants within a group. I refer to searchability as enabling the user to search for a specific element in a hierarchy by providing the elements name or part of it. For instance, if a user would be interested in viewing the streams of the source study participant #1933, the user would type 1933 (or 19) in a search box.

I have defined the following hierarchy:

-Stream – streams are the lowest level.
-Source – a source contains one ore more streams.
-Group – a group contains one or more sources.
-Study – a study contains one or more groups.

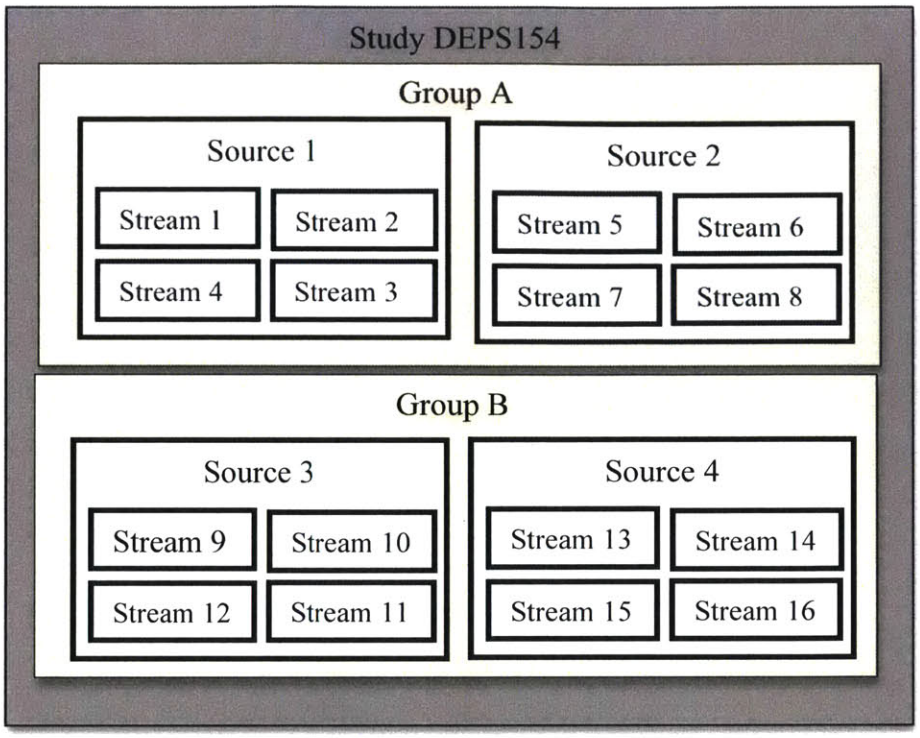


Figure 15. Hierarchical Information Architecture

Thus, a user could browse through (list in a specific order) all of the streams in a specific study, grouped by their respective groups, or search for a specific stream in a study.

4.3 User Interface Design and Wireframes

The main goal of the design process was to build a system that would achieve the quality of pliability. Lowgeren and Stolterman (Lowgren & Stolterman, 2007) define a set of information to be *pliable* to the user if it feels like a responsive material that can be manipulated in an almost tactile sense. Pliability may be achieved by "tight coupling", which was introduced by Ahlberg and Shneiderman (Ahlberg & Shneiderman, 1994). The main idea behind *tight coupling* is to minimize the distance between user intentions, user actions, and the effects of those actions. This goal becomes especially challenging when dealing with a large dataset as the computational complexity may increase the distance between the user intentions, actions and their effects. Thus, we utilized several methods to achieve tighter coupling:

1. Maintaining a low-resolution version of each stream enables fast retrieval and visualization.
2. After applying operators to the dataset, visualize representative samples of the result instead of the result set. This increases the speed at which the user can view results of operators.
3. Natural language querying facilitates easy access to the data without the normally imposed constraints of schema based querying; the user is not required to specific where to get the data from (which table) but rather only what data they are interested in retrieving.

Various stakeholders were interviewed as part of the design process. After gathering their requirements and wishes, a user interface wireframe was created. This wireframe is a skeleton representation of the UI. The wireframe illustrates the layout of each page in the web application; this includes interface elements and navigational elements, and how they work together (Garrett, 2010). The wireframe is usually low resolution, and lacks typographic style, color, or graphics, as its focus is behavior, functionality and, location and priority of content. Each screen was designed with the needs of each of the personas in mind. The ultimate design goal was aiming to maintain simplicity while providing each of the personas with the feature set that would enable them to achieve their goals by best utilizing the system. In each of the following sub-sections, I will review the functionality of each interface element along with its design goals.

4.3.1 Login Screen

The login screen is the main entry point to the system. It enables the user to create a new account, or to login with an existing account using a number of authentication providers (Google, Facebook, and LinkedIn). Once the user is authenticated and logged in, they can perform operations depending on their credentials. A user will remain logged in until they either logoff or their login session will expire after a preset default time.

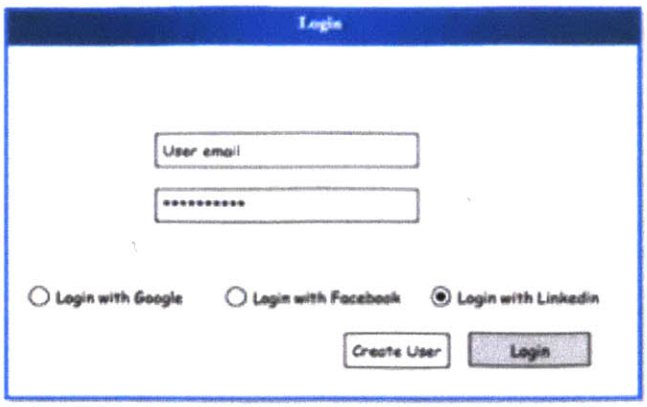


Figure 16. Login screen wireframe

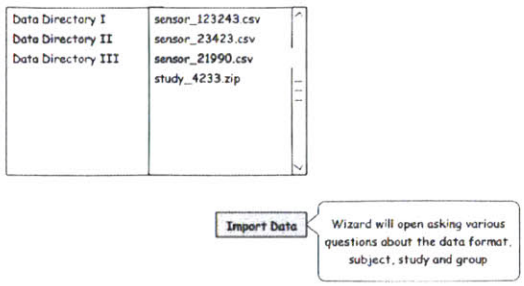


Figure 17. Data upload wireframe

4.3.2 Data Upload screen

The data upload screen enables the user to perform batch imports of data into the system. The user can either select individual sensor files or archives that contain multiple sensor files.

4.3.3 Study Creation

A study is a logical grouping of groups, participants (sources), or streams. The user can create new studies and assign streams to those studies. Any of the lower hierarchies could be assigned to a study – one or more groups, one or more participant, or one or more streams. After a study is created users can query or analyze data on a study basis.

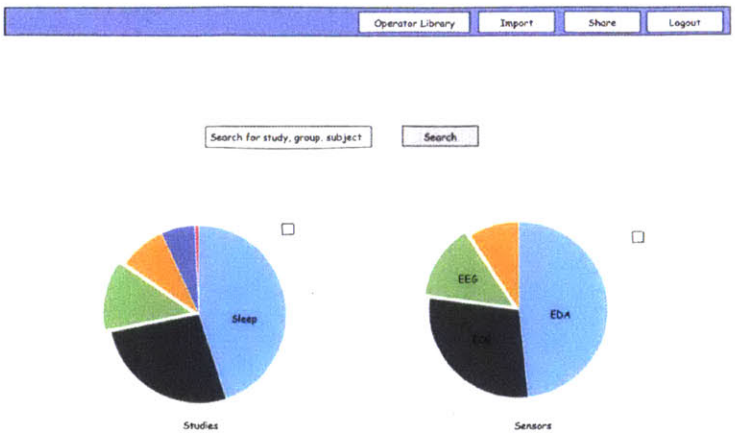


Figure 19. Dashboard view wireframe

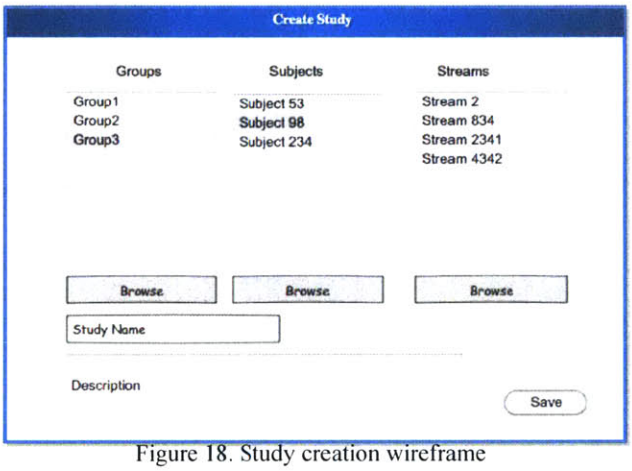


Figure 18. Study creation wireframe

4.3.4 Dashboard View

The dashboard provides a high level breakdown of the various datasets currently in the system. It may be used by a stakeholder interested in the current high-level state of the data collection for a study. Examples of such metrics are amount of data collected for each study, or breakdown of the amount of collected data on a sensor type basis (EDA, Temperature, Heart-rate, etc).

4.3.5 Group Creation

Groups are created similarly to studies. A group represents a logical grouping of participants or streams. Any of the lower hierarchies (participants and streams) may be assigned to a group. After a group is created users can query or analyze data based on that group. In addition, users can assign the group to a specific study.

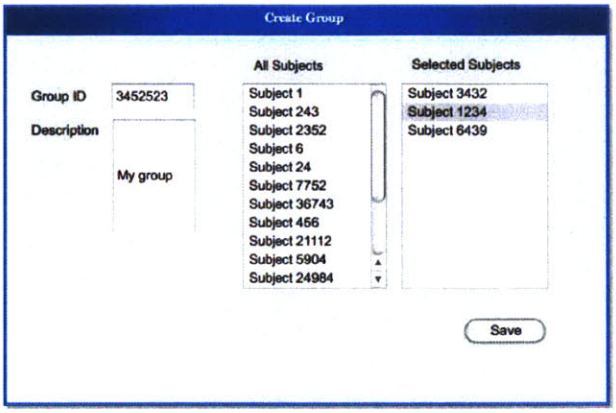


Figure 20. Group creation wireframe

4.3.6 Group View

The group view enables the user to view which participants (and their respective streams) are assigned to a group.

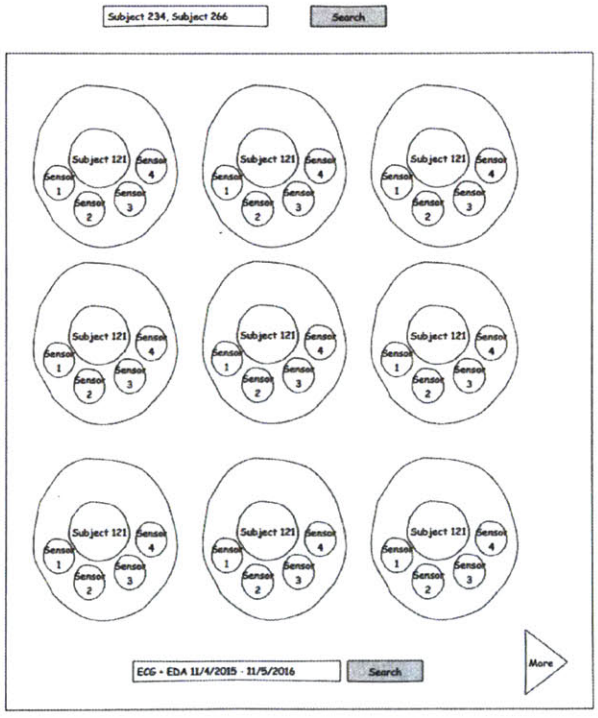


Figure 21. Group view wireframe

4.3.7 Study View

The study view enables the user to view all of the studies in the system (that he is authorized to view) and information about each study: number of participants, sensors, dataset size, etc.

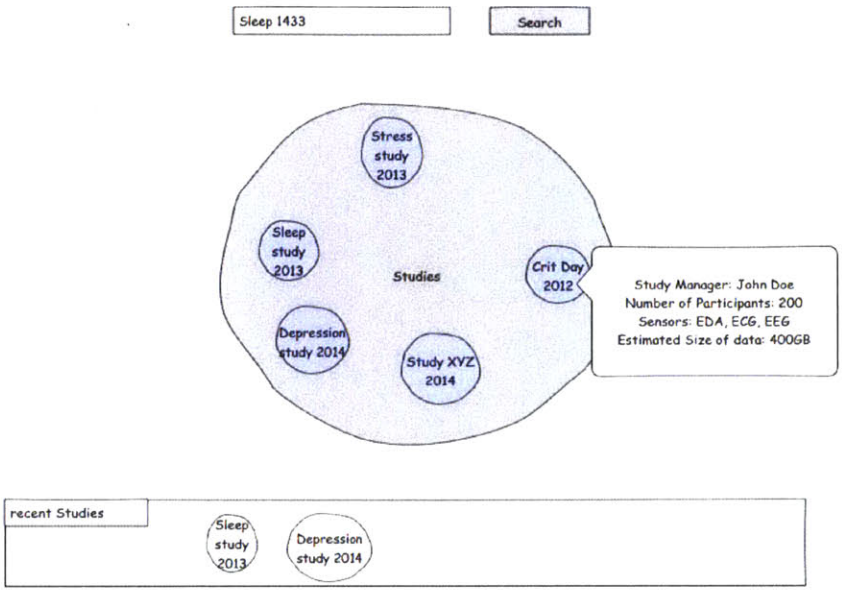


Figure 22. Study view wireframe

4.3.8 Stream View

The stream view enables users to search for streams based on specified criteria. Users can search streams based on participant IDs or parts of them, date ranges, sensor types, Study IDs or group IDs. The results contain a low-resolution plot of the stream and meta-data information such as sample-rate, number of samples, and start and end dates. The stream view also displays distribution data of the queried streams (grouped by sensor type): maximum, mean, and minimum. This view is the main entry point for interacting with the data. The design goal was to create an interface that enables the users to explore the data without being aware as to which streams are stored in the system. Search queries should complete within no more more than several seconds to enable *tight coupling* between the user and the data.

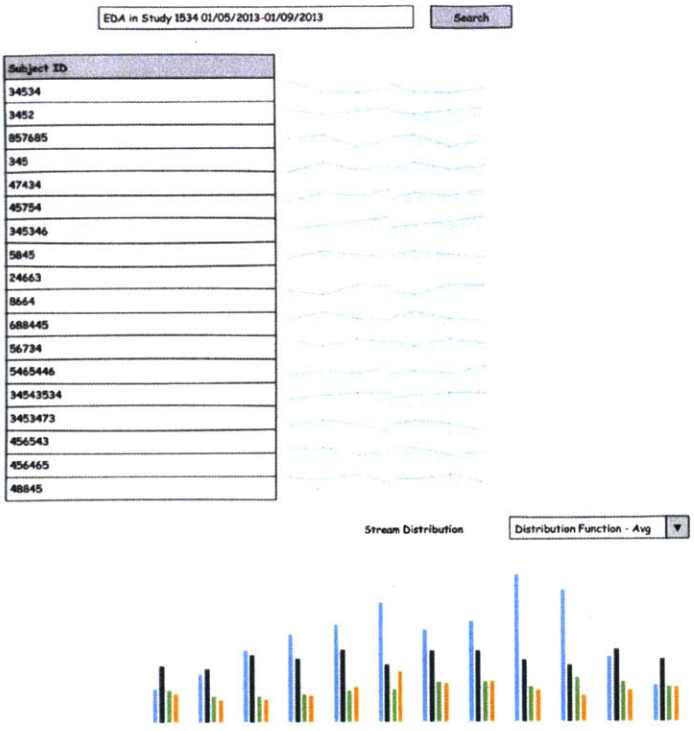


Figure 23. Stream view wireframe

4.3.9 Flow view

Users can create sequence of operators by chaining the output of one operator and the input of another operators into a flow. The flow view enables users to create new flows or edit existing ones. Users can drag operators from the operator library onto the canvas and draw connections between the operators.

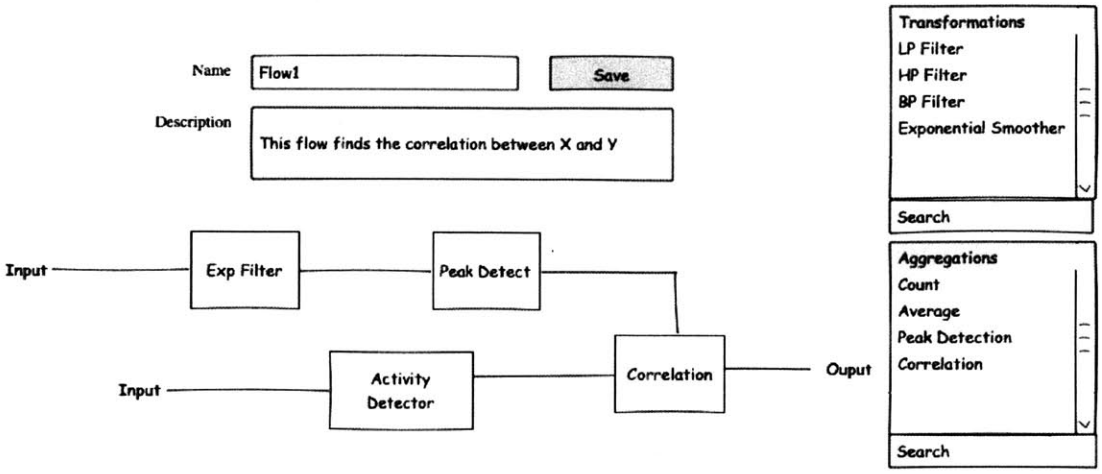


Figure 24. Flow view wireframe

4.3.10 Analytics View

The analytics view enables users to interact with streams by applying operators to them and viewing the results. The results can serve as input for additional operations. In addition, operators may be re-applied to the original data set numerous times, in order to compare between iterations with various parameters. Users can load multiple groups of streams into the analytics view and apply the operators to those groups. This design of this interface represents one of the main challenges; what elements should an interface that enables users to interact with massive data sets contain? What part of the process should be visualized?

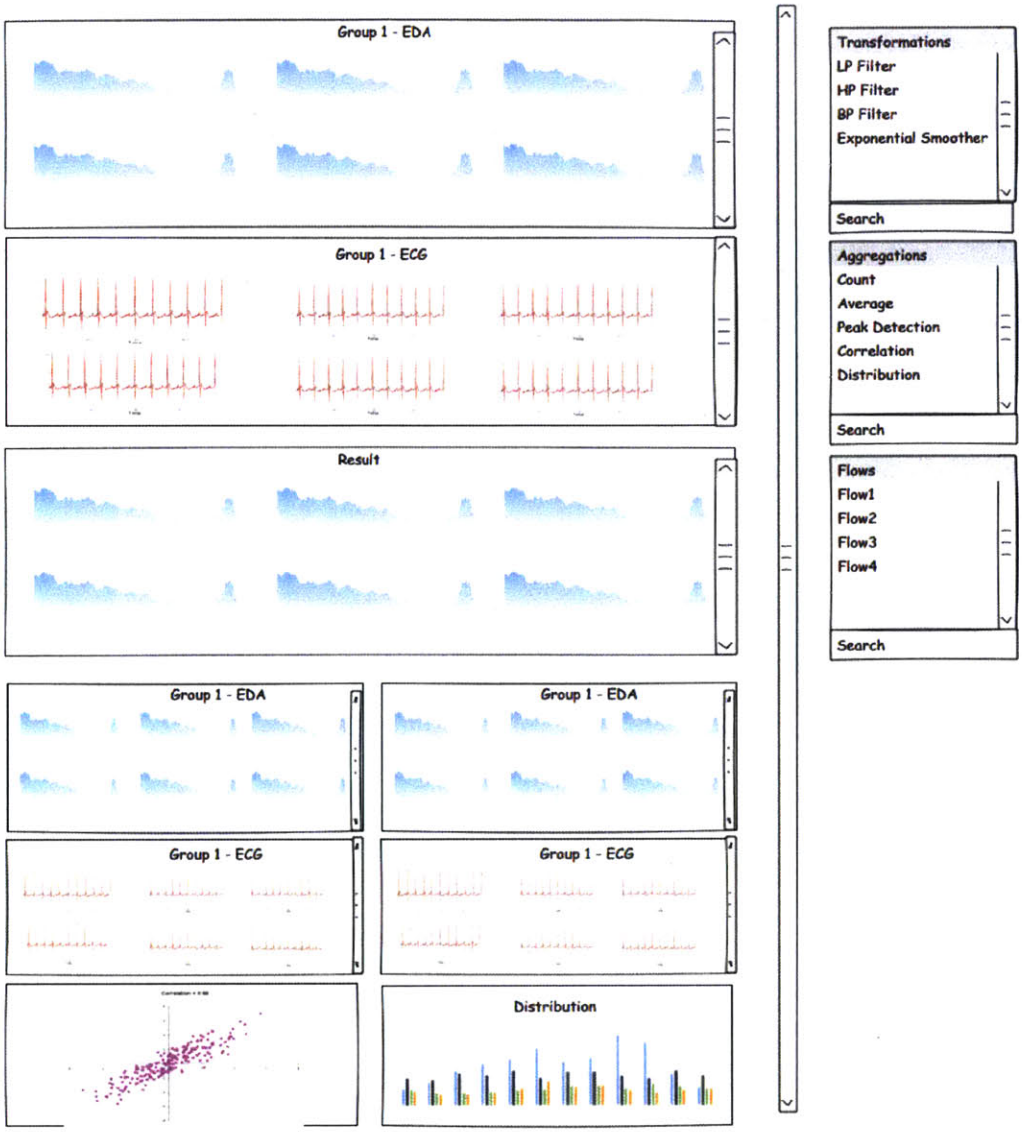


Figure 25. Analytics view wireframe

4.3.11 Code Editor View

The code editor provides the user with an interface to create new operators and edit existing ones. The user can select which language the operator is written as well as other operator parameters such as input type, output type, and user input variables.

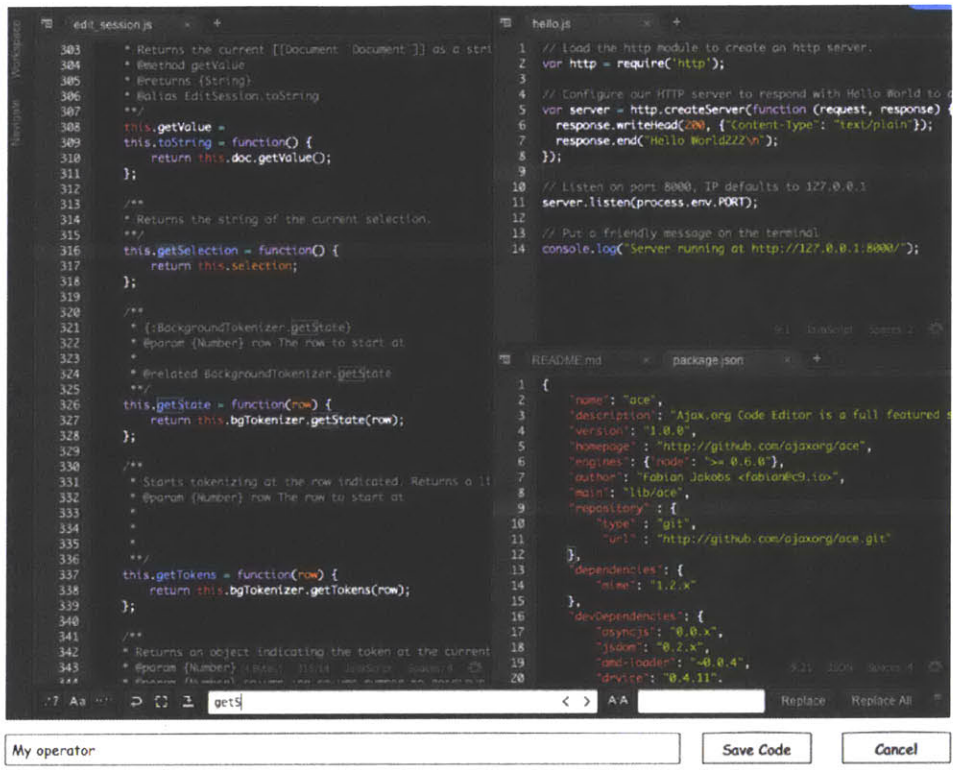


Figure 26. Code view wireframe

4.4 User Study

The purpose of the user study was to evaluate a prototype version of the analytics interface, as this was one of the complex interaction models in the system. General guidelines were given to all participants. They were told that this was a low fidelity prototype; its main goal is to test the interaction with the UI. For instance – search results are prepopulated on the screen. In addition, they were told that not all data would be present, and not all fields would be populated. Whenever that happened, we would let the user know. Three users participated in this study. In the first part of the study (questions 1-4) we asked the users open-ended questions to try and assess their backgrounds. In the second part, we gave them a list of use case scenarios and for each of them the user was required to navigate the controls of the user interface correctly. We wanted to test how intuitive the UI was, and the correctness of the UI layout. Below is a summary of the users interactions and responses.

1. ***What is you role here at the media lab?***

   P1 – grad student

   P2 – grad student

   P3 – researcher work on ML and physiological signals

2. ***How Often will you analyze a dataset?***

   P1 - once a day

   P2 – once a day

   P3 – once a day at least. Sometimes more.

3. ***How many passes would you have to make at the data before you are happy with it (levels of iterations and refinement)***

   P1 – Quit a few

   P2 – Dependent highly on the study and the data (multiple times)

   P3 – Many dependent on the dataset. As many as time allows.

4. ***What is your primary objective when analyzing data?***

P1 - Probably to compute features from it and To distill an aggregate a large part of the data to a few metrics

P2 – to gain new insights and to prove hypotheses. To look at things beyond their face value

P3 – I want to gather insights from the data, correlations, patterns that are associated with specific hypothesis.

5. ***How would you load the results on the screen for analysis?***

P1 – user hits the analysis button

P2 – user hits the analysis button (after wondering aloud what they should do)

P3 – user tries to understand what loading the results means, then clicks the analysis button

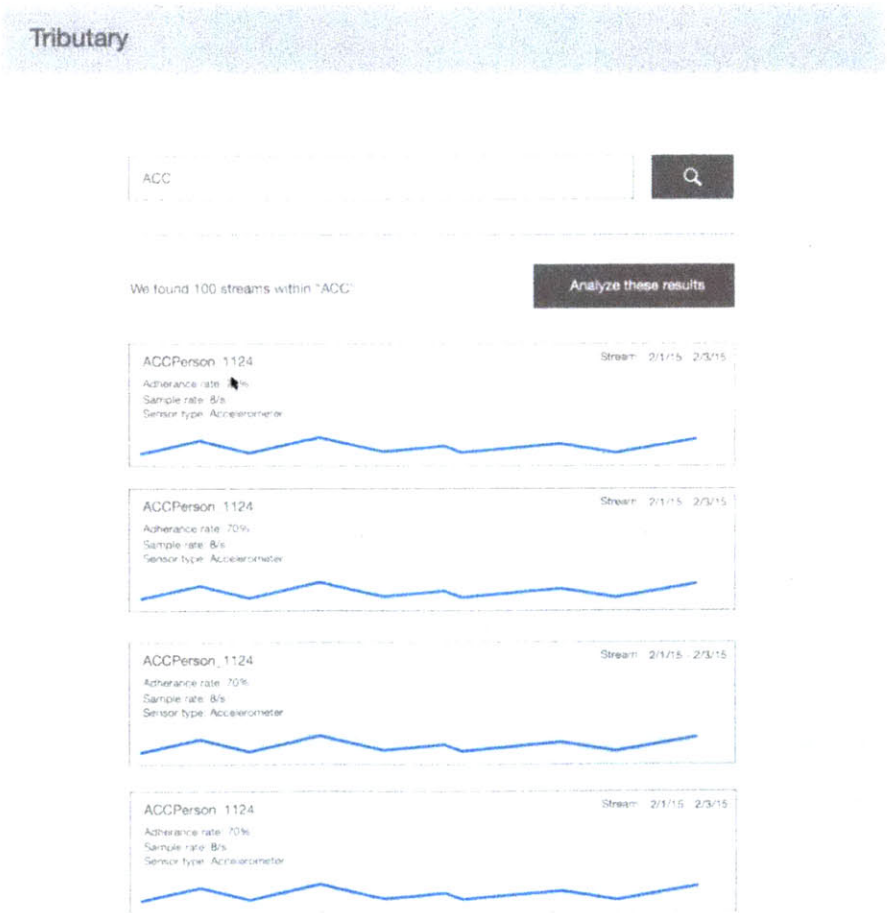


Figure 27. Users were asked to load the results for analysis (Question 5)

6. **How would you apply an operation to that data ?**

P1 – user clicks apply operation button

P2 – user clicks apply operation button

P3 – user clicks apply operation button

7. ***You want to apply another operation to the same data, and compare between the results, how would you do that ?***

P1 - User clicks apply operator to the first node again, and then hits the compare button on the result node

P2 - User clicks apply operator to the first node again, and then hits the compare button on the result node. When user tries to close the dialog, they hit the delete button instead of the close dialog button.

P3 – User clicks apply operator to the first node, and then hits compare. When user tries to close the dialog, they hit the delete button instead of the close dialog button.

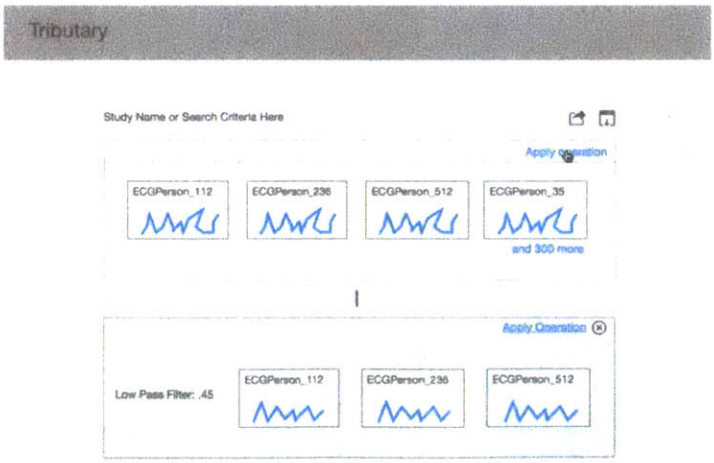


Figure 28. Users were asked to apply an operation to the data (question 6)

Figure 29. User applies an additional operation to the root node and the resulting nodes are stacked (question 7)

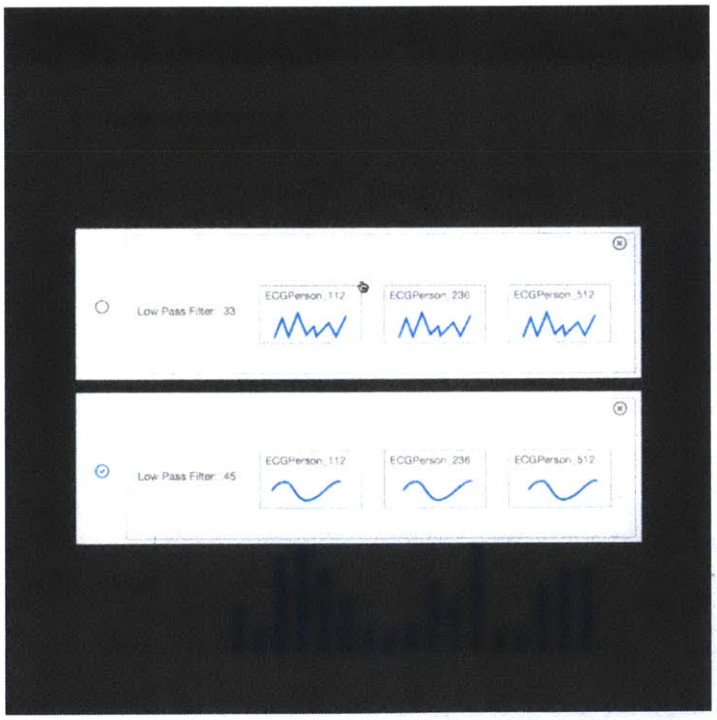


Figure 30. Users were asked to compare between results of two operations (question 7)

9. Within the comparison overlay – how would you select your previous result in order to continue the computation with it?

P1 – User clicks the select result radio button

P2 – User clicks the select result radio button

P3 – User clock the select result radio button

8. How would you apply an average operator to the current set of results?

P1 – User click the apply operation button on the last node in the tree

P2 – User clicks the apply operation button on the last node in the tree

P3 - User clicks the apply operation button on the last node in the tree

Figure 31. User were asked to select the previous result in order to continue the analysis with it (question 8)

10. Apply a new operation to your starting data

P1 – clicks correctly

P2 – Clicks correctly

P3 – clicks correctly

11. You aren't too happy with that and you would like to create a new version from you original data. How would you do that?

P1 – clicks correctly

P2 – Clicks correctly

P3 – clicks correctly

12. You would like to delete that stack. How would you do that?

P1 – clicks correctly

P2 – has trouble finding the delete button

P3 – clicks correctly

13. One of the streams seem a bit off, how would you view the full stream in this version of the prototype ?

P1 – clicks correctly

P2 – Clicks correctly

P3 – clicks correctly

14. It wasn't a big issue, apply an operator to these streams

P1 – clicks correctly

P2 – Clicks correctly

P3 – clicks correctly

15. Apply an operator to the result of the previous operator.

P1 – clicks correctly

P2 – Clicks correctly

P3 – clicks correctly

Post study questions:

What features did you find most valuable?

P1 – that you could apply a sequence of successive operations to the data. Also being able to apply an operation to 300 streams at once

P2 – that I could see the flow of operations, and that I could easily see the data,

P3 – being able to see the previous iterations. Being able to zoom in and out on the signals

Why?

P1 - It was easier to see, and I wouldn't have to create my own scripts, import the data, etc

P2 – I could do all that without doing very much work

P3 – its important to be able to look at different time granularities.

What did you find confusing?

P1 – The analyze results button – I wasn't sure which results would be loaded. I was sure it would just be the first one that I clicked.

P2 – originally the apply operation button was confusing. After I realized that it was a tree and that I could apply an operation to any node on any level, things became clear.

Figure 32. Users were asked to apply an average operator the current set of results (question 9)

After completing the study it became clear that the design was a viable approach to the analytics user interface and that we could move forward with the implementation.

Design must reflect the practical and aesthetic in business but above all... good design must primarily serve people.

Thomas J. Watson

5 System Design and Architecture

When designing a system several important questions come to mind. First and foremost:

"What are the goals that the user is trying to achieve?"

"What is the most efficient path to achieve those goals?"

The second question entails much complexity. "Efficiency" is defined as the ratio of the useful work performed by a machine or in a process to the total energy expended or heat taken in. Therefore, the second goal could be loosely translated as providing the user with a path that would be shorter than others. This statement begs the question: "short in what way"? Would this be the "shortest length of time"? Or the "smallest number of steps"?

They are not necessarily one in the same. Let's examine the case of a researcher who is interested in examining a sensor dataset collected as part of a study.

5.1 The Value of the Human in the Loop

Before being able to ask questions of the data (testing various hypotheses) the user may want to assess the quality of collected data. Namely: was the data the study initially set out to collect – successfully collected and stored as intended. If the amount of collected data is small this is a trivial task, which can in some cases be performed manually or by using a single computer. However, in case of a large-scale data set containing millions or billions of samples this is a daunting challenge. In order to execute this task efficiently, the user may choose to fully automate it. At first glance, this may appear to be the most efficient solution in both number of steps and length of time as every iteration is fully automatic, assuming the implementation is efficient as well. However, a new challenge is introduced. In order to successfully automate a task, the user has to possess knowledge of a model that represents the underlying physical system

and be fully aware of its behavior and constraints. In cases of well-studied areas, the users can obtain the knowledge that will be sufficient to automate the task at hand. In fields that are less understood either because they are novel, or due to the fact that they are less studied, using a fully automated approach may in fact be detrimental to the goal at hand. The knowledge required to distinguish between a "clean" and "dirty" signal may be constantly evolving.

The user will need to modify and re-execute the automation each time new information is learned in order to try to achieve a global optimum. Perhaps a different approach will prove to be more efficient.

The user may apply a semi-automated or "human-in-the-loop" approach. Human-in-the-loop (HTL) is defined as a simulation that requires human interaction (Karwowski, 2006). HITL allows for the identification of problems and requirements that may not be easily identified by other means of simulation, It enables the operator to change the outcome of a process as well as enables the operator to acquire knowledge of how a new process may impact an event. By utilizing this approach, the user can utilize new information as new samples are being reviewed, update their "mental model" of the system and provide inputs to the system for the next steps.

The challenge then becomes: what would be the most efficient method for a user to interact with a large-scale data set. In his seminal paper "The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizaiton" (Shneiderman, 1996), Schneiderman defines the visual seeking mantra: Overview first, zoom and filter, details on demand. This mantra served as one of the basic principles in the implementation of the Tributary system. When dealing with billions of samples, it is necessary to provide the user with tools to effectively navigate through the data.

5.2 Guiding design principles

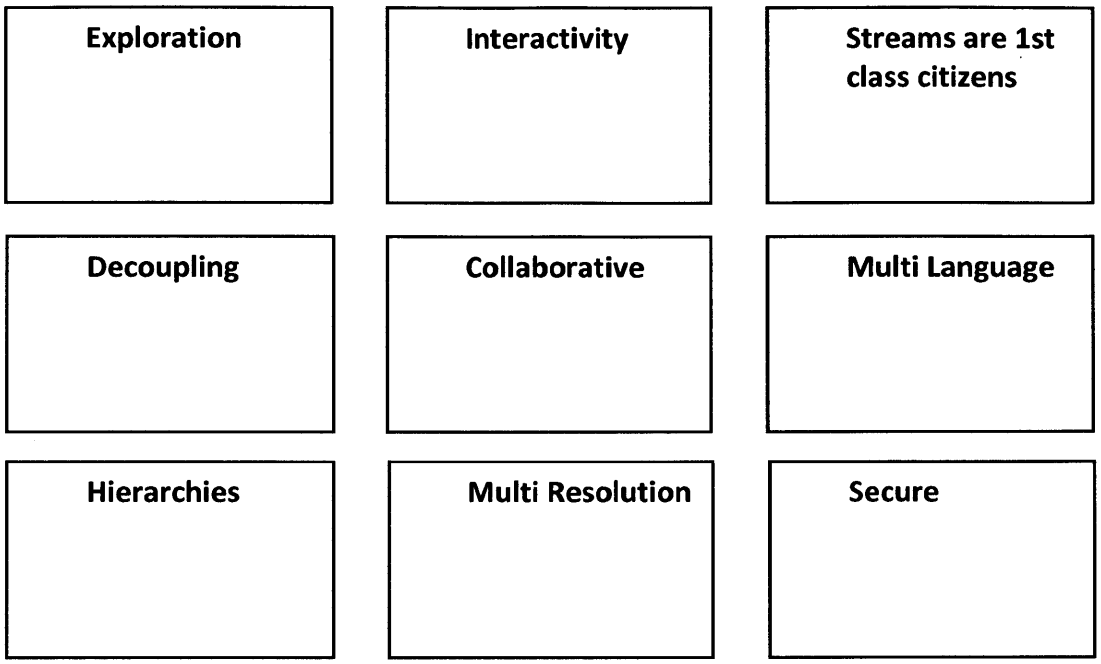


Figure 33. Design principles

The Tributary system was design with the following goals in mind:

1. **Exploration** – data may be accumulated from multiple sources. Some of the data may be uploaded manually, and some may be streamed in real-time. The person importing data into the system may not necessarily be the one to analyze the data. Thus, a user should have the ability to explore the data – be able to view the dataset without knowing exactly what it is composed of. For instance, rather than search for a particular study participant, the user can view all of the data streams according to each participant within a study.
2. **Interactivity** – the system should enable the user to interact with the data: search for data based on various criteria, apply operations to the data and view the results. The user can make decisions on which parameters to modify in order to achieve the required results.
3. **Streams as 1st class citizens** – data steams are entities that can be interacted with and operated on. This is in contrast with traditional data science tools that view numerical

data as 1st class citizens (Scott, 2009) and require the user to implement higher-level abstractions.

4. **Decoupling** – the data structures, their formats as well as their storage location are completely decoupled from the analytics operators. From an operator standpoint, the data are placed in a continuous array.
5. **Collaborative** – a user can share data and operators among users.
6. **Multi Language** – operators can be coded in multiple programming languages. A user does not have to learn a new language in order to use the system.
7. **Hierarchies** – all of the data are organized in hierarchies. Users can search the entire space of hierarchies.
8. **Multi Resolution** – streams can be stored in any sample rate and the user can view streams at any resolution (from minutes to years).
9. **Secure** – access control enables only authorized users to view specific data. The data owner can grant access rights to additional users. Access control is maintained at the stream level.

5.3 Nomenclature

Stream – a stream is a data time series. It contains multiple samples ordered by time. A stream can be either univariate or multivariate and contain either numerical data or textual data. A univariate numerical stream is of type 'Double', a multivariate numerical stream is of type 'MultiDouble' and similarly for textual streams: 'Text' and 'MultiText'.

Source – a source is a grouping of streams that usually identifies the origin of the data. For example – a person wearing several wearable sensors is a source and each of the sensors generates a stream. Or a sensor node recording environmental signals is also a source.

Group – a collection of sources or streams and is used to identify a category. This can be a study cohort, a geographical region, etc.

Study – a collection of one or more groups. The study is used mainly for administrative purposes. It enables study managers to query and view data as part of a study that it was collected for.

Operator - a construct that behaves like a function, which operates on streams and user input parameters. For instance, an operator may receive as input a temperature stream and output the average temperature.

5.4 Architecture overview

The Tributary system is composed of multiple services. Each of these services is a standalone component designed to provide a set of functionalities. They may reside on one or more physical machines to support high availability and scalability. Image n shows the various architectural components and their hierarchy. In this section, I will describe each of the services design and functionality.

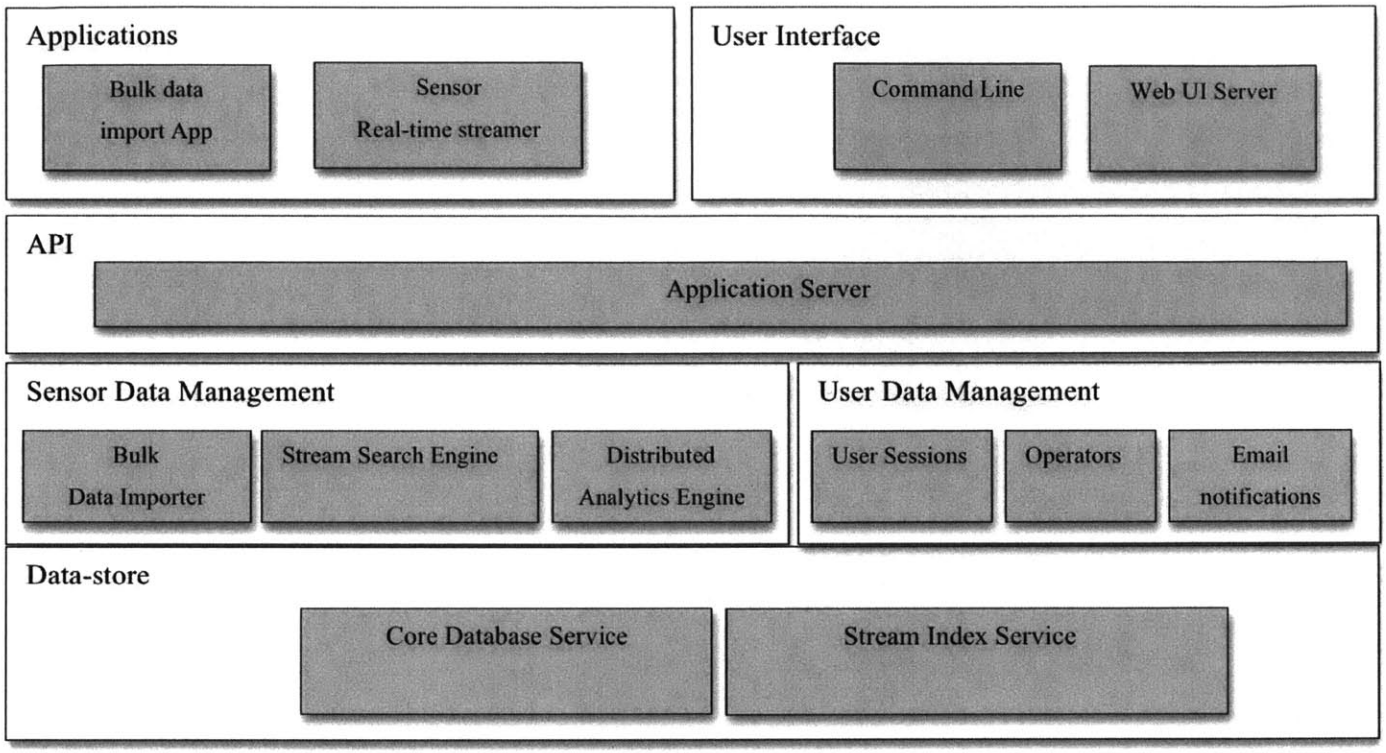


Figure 34. System Architecture Block Diagram

5.4.1 Core Database Service

The core database service is responsible for storing the time series data. Each data point is stored along with the timestamp at which it was sampled. All time stamps are stored in millisecond resolution. The service provides applications with an interface to query time series by source-id, stream-id and date/time ranges. Each stream is divided into blocks of n samples and each block is stored on a machine (node) in the cluster.

The requirements for the core database service were as follows:

1. All the streams of a specific source-id should be distributed into blocks and each block should correlate with a range of timestamps. The data should be distributed as evenly as possible across the nodes in order to prevent host spots.

2. All the blocks for a specified range that belong to a specific source-id should be collocated on the same physical node. As in many cases, analysis is done for a specific source id across several sensor streams, having them all located on the same node would reduce the network IO.

3. It should be possible to query a stream by: source-id, stream-id and a range of dates.

The service is currently implemented using Apache Cassandra (Lakshman & Malik, 2010) – an open source distributed wide column key-value data store that is designed to handle large amounts of data across many commodity servers. Cassandra provides high availability, elasticity and no single point of failure. In addition, due to its architecture, Cassandra is able to achieve high throughput, which fulfills an important requirement for a large-scale data analytics system. Cassandra's data model is a partitioned row store with tunable consistency. Rows are organized into tables; the first component of a table's primary key is the partition key; within a partition, rows are clustered by the remaining columns of the key. Other columns may be indexed separately from the primary key.

In order to satisfy the above requirements, the core database service utilizes the following schema: the partition key is a composite of the source-id and a minute resolution timestamp. All samples from the same source-id, sampled at the same minute will have the same partition-key and as a result be located on the same node in the cluster. This facilitates sensor fusion as all the samples of a specific time for a single source will be located on the same machine and thus will not have to be transmitted over the network during computation. Partitioning the samples on a minute-by-minute basis was done in order to limit the database row maximum sizes, as large rows require additional memory and computational resources.

The row-clustering key is a composite of source-id, stream-id, and timestamp in milliseconds. This enables the user to query the system for a sample either based on source-id, or on a combination of source-id and stream-id, or on a combination of source-id, stream-id and timestamp.

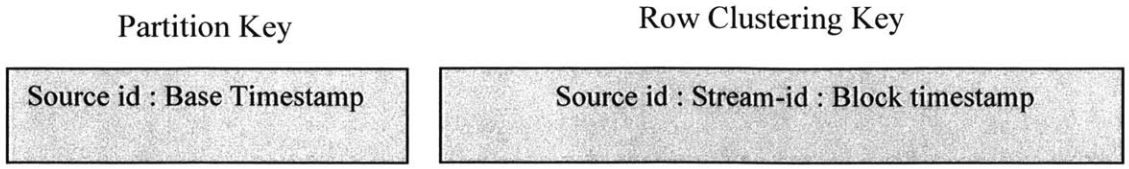


Figure 35. Cassandra database schema

5.4.2 Stream Index Service

The stream index service holds all of the stream meta-data. It provides the user with a method to query the system and retrieve meta-data regarding a specific stream or group of streams. A meta-data record contains the following information:

Source id – the stream source-id

Stream id – the stream-id

Group id – a stream can be associated with a group. This can be used to designate a stream with a study cohort, a location of collocated streams, etc.

Study id – a stream can be associated with a study.

Stream type – a stream can be one of the following types:

-Float: each sample is a single floating-point number

-Multi-float: used for multi-modal data. Each sample is composed of multiple floating-point numbers. For example: GPS data consists of latitude and longitude, and Accelerometer data consists of X, Y, and Z

-Text: each sample is a text value

-Binary: each sample is a collection of bytes.

Modalities - in case of multi modal streams, signifies the number of modalities.

Sensor type id – signifies the type of sensor. For example: accelerometer, temperature, humidity, EDA, ECG, etc.

Start date – the start date of the stream in milliseconds since the epoch.

End date – the end data of the stream in milliseconds since the epoch.

Number of samples – the number of samples in this stream

Sample rate – the sample rate in Hz

Max – the maximum sample in this stream

Min – the minimum sample in this stream

Average – the average of all samples in this stream

Stream owner – the owner of this stream. The user-id who uploaded or created the stream

Stream access list – a list of users who are allowed to access this stream.

Stream segments – a list of tuples that denote the segments of each stream. A stream may have one or more segments. A segment is a continuous uninterrupted recording of samples. A new segment is created when a sensor starts recording samples. The tuple's first element is the start time of the segment in milliseconds since the epoch, the second element is the end time of the segment, and the third element is the number of samples in the segment. Note that the segment length can be calculated on-the-fly using the segment start, end and the stream sample rate. It is pre-calculated in order to save computation time.

Gaps – a list of tuples that denote the time in milliseconds between the segments.

Completeness – this designates how complete the stream is. It is a ratio of the excepted number of samples and the actual number of samples. In case of a wearable sensor this can be used to determine the adherence of a participant.

Hardware Id – used to denote the sensor manufacturer, and device version number.

When a user queries the index, only streams that contain that user's user-id in the stream access list will be returned. By default that access list is populated with the stream owner's user-id. A user can query the index by source-id, stream-id, and date ranges, or a combination of any of them. The user can also limit the number of results in order to reduce query execution time.

5.4.3 Stream Search Engine

The stream search engine receives queries from the user, parses them and retrieves the streams that comply with the search criteria from the stream index. The parser algorithms is as follows:

1. The search engine creates tokens from the search query; the tokens are delimited by the space character.
1. Each of the tokens are compared against the following list of keywords:
   [before, after, between, or, and, group, study] in both upper case and lowercase.
3. If a keyword is found, the following or previous token will be analyzed and a search criterion will be extracted according to the following rules:
   a. For 'before' or 'after' the following token will be parsed as a date.
   b. For 'between' the preceding and following tokens will be parsed as dates.
   c. For 'or' and 'and' the preceding and following tokens will be used to match stream-ids and source –ids. A Boolean query will be generated.
   d. For 'group' or 'study' the following token will be used to search for all the streams that belong to a group or study that contain that token in their name.

5.4.3.1 Stream Index Query Language

The following query syntax is supported by the stream index service:

Query based on source id:

It is possible to query all sensors of a specific source id by entering the full source id or part of it. For example: 8110 will result in all of the sensors for source-id 8110 will result in all sensors for nodes containing 811 in their source-ids. The user can also combine a sensor-id to a source-id in order to return only data for that sensor. For example: 8110 TEMP will return the temperature for source 8110 811 TEMP will return temperature for all streams with 811 as part of their source-id. The user can also query multiple source-ids using the 'or' keywords for example: 8110 or 8114 will return all streams for sources 8110 and 8114. It is also possible to query several source-ids for a specific stream-id by using the and keyword. For example: 8110 8114 8115 and TEMP will return the TEMP streams for the 3 source 8110, 8114 and 8115.

Query based on stream id:

The user can query a specific sensor across all sources by typing the sensor-id for example: ACC will return the accelerometer stream for all sources. It is also possible to query multiple sensors by using the or keyword: BATT or TEMP

Query based on date ranges:

The user can use the *before, after, and between* keywords to limit the dates of a query. The format of the date is one of the following:

- yy/mm/dd for example: 14/02/25
- yyyy/mm/dd for example: 2014/02/13
- yy/mm/dd HH:MM for example: 13/11/22 17:43

Example queries: TEMP before 15/04/21

8110 TEMP after 15/04/21

Query and limit the number of results:

It is possible to limit the query execution time by returning a maximum of n results by querying in combination with the limit keyword. For example: TEMP limit 10 will return the first 10 temperature streams.

All of the query results are always sorted based on source-id.

The stream query language enables the user the flexibility of retrieving streams according to a wide variety of criteria. The stream index database is currently implemented using MongoDB as the data storage layer. MongoDB is a NoSQL document oriented database with dynamic schemas. Each stream meta-data record is represented as a BSON document, which is a binary representation of JSON and each field is represented as a field value pair. The query engine is implemented by utilizing the MongoDB query interface that uses a JSON-like query language.

5.4.4 The Application Server

The Application server is the main interface of the system and provides a RESTful interface to user applications. REST or Representational State Transfer REST is an architecture style for designing networked applications. It was introduced by Roy Fielding, one of the main authors of the HTTP interface in his PhD dissertation (Fielding, 2000). REST relies on a stateless, client-server, cacheable communications protocol and usually utilizes HTTP as the transfer layer. The main benefits of this type of approach are:

1. Simplicity – each interface is well defined. All elements in the system are resources that are self-descriptive and addressable via Uniform Resource Identifiers (URIs).
2. Decoupling – the applications are decoupled from the API. The enables modifiability of components to meet changing needs even while the application is currently executing.
3. Transparency –as the protocol is text based it enables visibility of communication between components by various agents
4. Portability - applications can be implemented on any platform.
5. Reliability – the protocol is implemented over HTTP and thus is resistant to failure at the system level in the presence of failures within components, connectors, or data.

In order to interact with the API server, the user application must authenticate itself by calling the /login endpoint and providing the necessary credentials. After calling /login, the application is authenticated. The application receives a token that can be used by the user to access the streams with the relevant permissions. This token is valid for a configurable amount of time or until the application calls the /logout endpoint.

The REST endpoints are described in detail in Appendix D.

5.4.5 Stream Low-Resolution Cache Service

The stream low-resolution cache service (SLRC) enables the user to retrieve a decimated (lower sample rate) version of each stream and is useful when visualizing the stream. As a stream can contain millions of samples, it is not feasible to retrieve all of them and display them

on a computer monitor. Using a low-resolution version of the stream holds additional advantages:

- Reduced disk IO, as the streams are stored in a database that resides on a disk.
- Reduced network IO, as the streams need to be transmitted to the user application over the network.

The low-resolution version is created when the stream is imported into the system and adheres to the overall design philosophy – import once, view and analyze multiple times. Thus, instead of subsampling on-the-fly each time the stream is retrieved, and being penalized multiple times, the stream is subsampled only once upon import.

Decimation is performed using a technique similar to the one described in (Lyons, 2010) Each segment is subsampled to contain 100 samples. When the segments are retrieved they are concatenated. If the resulting stream is larger than 1000 samples, it is subsampled on-the-fly to contain 1000 samples.

5.4.6 The Analytics Engine

The analytics engine distributes and parallelizes user operators and executes them on the data. The parallelization is twofold both on the stream level and the group of streams level: an operator is applied to multiple segments of the stream in parallel, and an operator is applied to multiple streams in parallel. This makes the system effective in dealing with both a small amount of large streams (such as years of recording) or a large group of streams.

The user loads streams in the analytics engine by executing various queries and deciding which streams to analyze. The loading operation reads the stream samples from the Core Database Service into the RAM of all the machines in the cluster. The streams are segmented into blocks (or chunks) and each block resides on a node. The segmentation is configurable by the user; a stream can be segmented into hours, days, weeks, or not segmented and loaded in its entirety into the RAM of a single node. Data loading is done in several stages: first, the streams are loaded into RAM using the original data store schema and then each stream is partitioned

into blocks. Each block is a key-value pair. The key is 3-tuple containing the source-id, stream-id and block start timestamp in milliseconds.

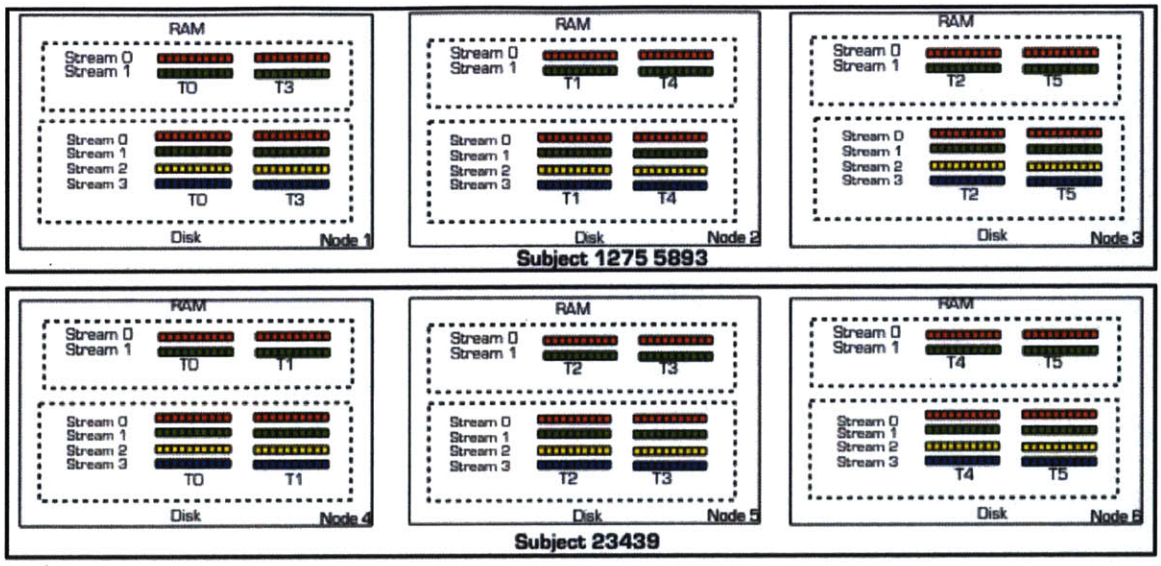


Figure 36. In this example figure, streams 0 and 1 are loaded into the RAM of 6 nodes in the cluster. Streams 0 and 1 of participant 1275-5893 are loaded into the RAM of Nodes 1-3 and streams 0 and 1 of participant 23439 are loaded into the RAM of nodes 4-6. Each streams is segmented into blocks (T0-T5). Locality is maintained as each segment is loaded from disk to RAM in the same node.

Users can execute operators on the loaded streams and the result is also retained in memory to serve as input for an additional operator execution. The analytics engine supports the four classes of execution described in chapter 3: transformations, aggregations, combinations, and selectors.

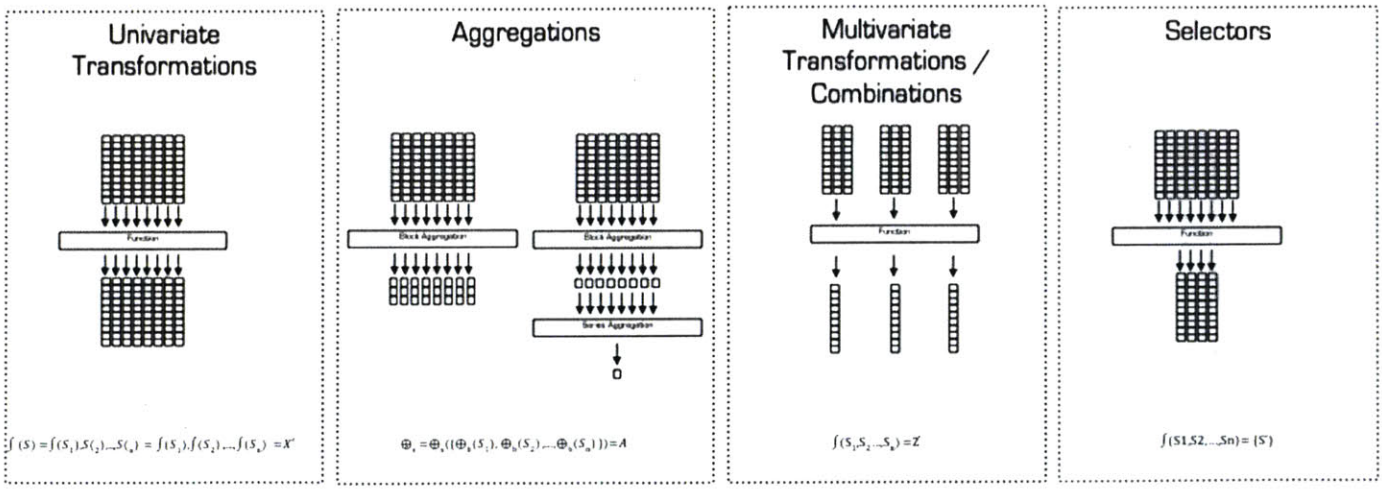


Figure 37. The four classes of operators supported by the analytics engine

5.4.6.1 Sessions

The analytic engine groups the streams into sessions. A session is a computational container that holds either the streams of a specific type of sensor (such as Accelerometer or EDA), or another criteria configured by the user (such as cohort). Transformations, Aggregations and Selections are applied to all streams in a specific session, while Multivariate transformations / combinations are applied to streams in several sessions.

Sessions are implemented as trees. The root node contains the streams that are loaded by the user query. It is always assigned an ID of 0. Applying an operator to a node will create a child node for that node with an ID that is randomly generated. Applying a different operator to the same node will create an additional child (or a sibling to the previous result). The user can compare between siblings, which is useful for comparing multiple iterations of the same operator, but with different parameters. For instance, the user applies a filter operator with a specific cutoff frequency to a node, and then applies the same filter with a different cutoff frequency to the same node. Both of the results are retained, enabling the user to compare between the various iterations. The user can delete a specific node from the computation tree.

Deleting a node will result in the deletion of its sub-tree as well. Deleting the root node will result in the deletion of the entire session.

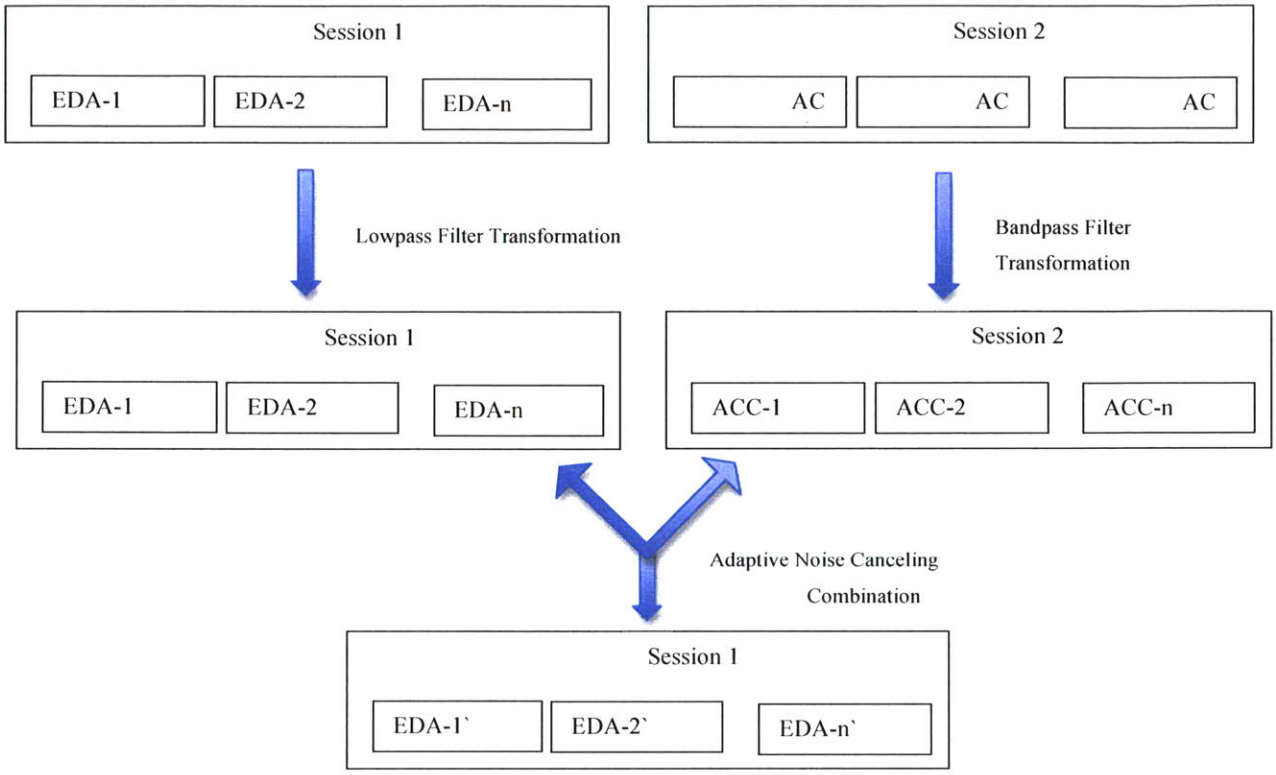


Figure 38. Session 1 contains all EDA streams and Session 2 contains all accelerometer streams. A low-pass filter is applied to session 1 and a band-pass filter is applied to session 2. An adaptive noise-canceling algorithm is used to filter out movement artifacts from the EDA signals by applying a combination to sessions 1 and 2. The results are stored in session 1.

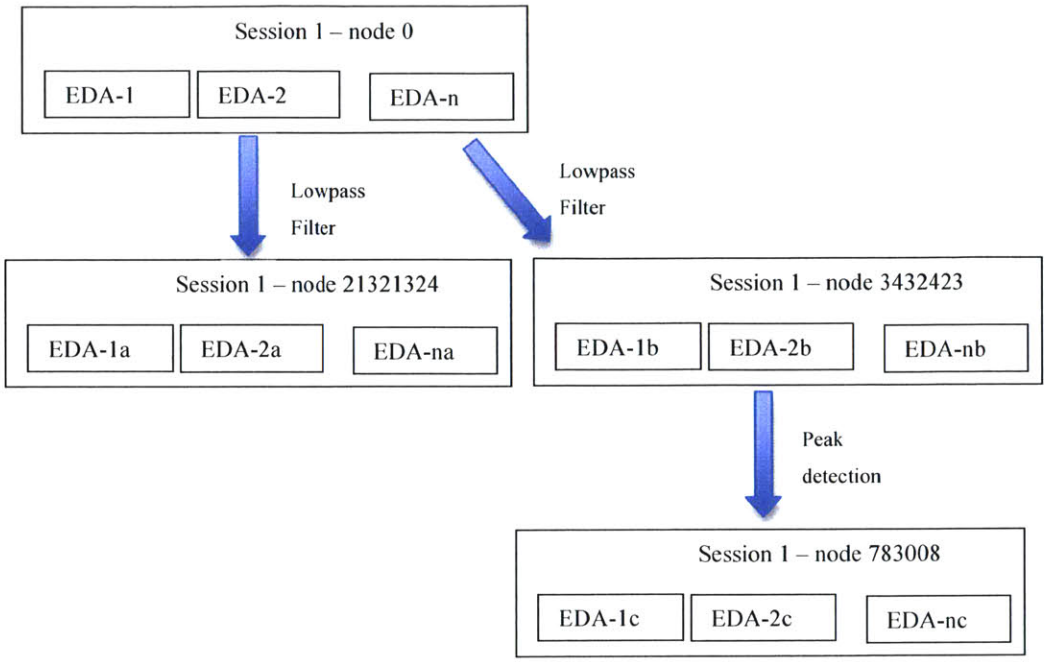


Figure 39. A session tree. The root node is generated when the user selects the streams to load for analysis. Subsequent nodes are generated when the user applies an operator to a node. Operators can be applied to any node at any point of time.

5.4.6.2 Analytics Engine Implementation

Currently, the Analytics Engine was implemented on top of Apache Spark (Zaharia et al., 2010). Spark is an open source cluster-computing framework originally developed in the AMPLab at University of California, Berkeley. The basic building block in Spark is a Resilient Distributed Dataset (RDD). An RDD is a set of records that are either values or key values that can be cached in RAM across the nodes of a cluster. However, these records do not maintain any temporal relationship, as there is no built-in support for time-series data in Spark. At the core of the Analytics Engine is a time-series implementation over Sparks native RDDs. This implementation provides facilities such as slicing a time-series, sub-sampling a time series, and various other statistical functions such as correlation, max, min, average, mean, as well as facilities for executing user defined operators on the data.

The time series is implemented as a key-value RDD. Each time series is segmented into blocks (or segments) where the block is an RDD value and the key is a tuple, which contains a millisecond resolution timestamp that signifies the block start time, the source-id and the stream-id. A block may either be univariate or multivariate. The number of samples per block depends on how the user decided to load the stream. For instance if a user decided to segment the streams in blocks of one hour, and the stream sample rate was 8Hz, then each block would contain – 1 hour * 60 minutes * 60 seconds * 8 samples/sec = 28,800 samples. The blocks are grouped into RDD partitions by the following formula:

$$Partition\text{-}number = block\text{-}start\text{-}time \ \% \ total\text{-}number\text{-}of\text{-}partitions$$

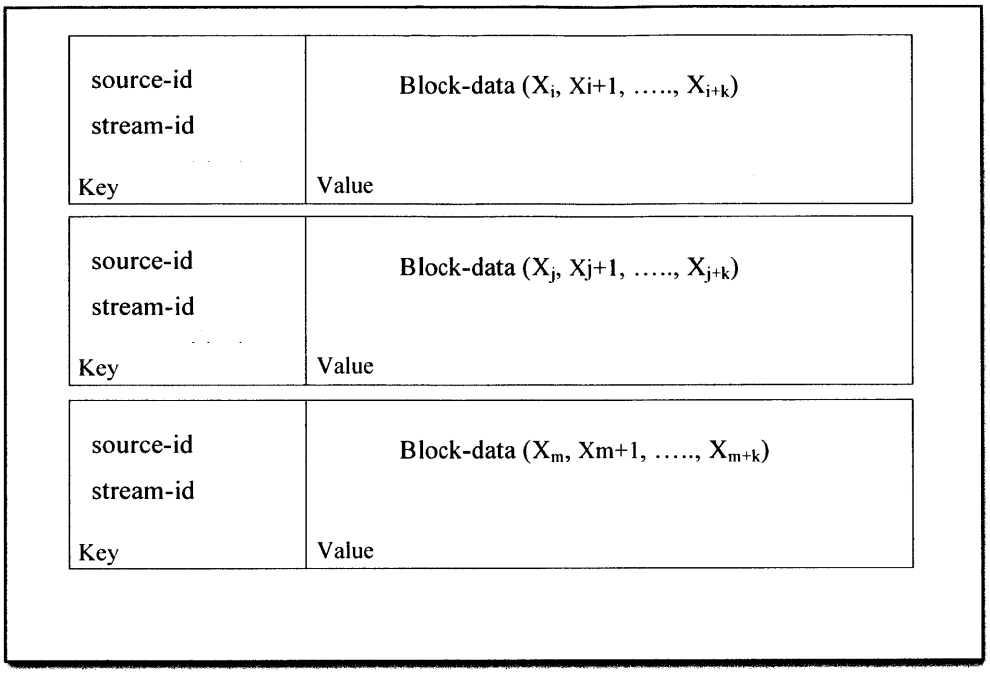


Figure 40. Spark RDD partition. Each partition resides on a physical node in the cluster, and contains several blocks of time-series. A blocks with the same start times will be placed in the same partition.

Transformations can be applied to each block separately and do not require shuffling of data across the network. However, combination operators (such as correlations) require access to multiple blocks in tandem and require that blocks be transported between nodes. This is achieved by grouping blocks with the same block start-time into the same partition.

5.4.6.3 Analytics Engine Operators

The analytics engine provides the user with a set of built-in operators that can be executed on the streams of data. In addition, users can implement their own operators in a variety of programming languages. In the current implementation, Python, Java and Scala are supported. These operators are building blocks that enable the user to design their own analytics pipeline. The output of each operator serves as the input of the next operator that is applied. In addition, the user can apply multiple operators to the same input and compare between them. Operators have input parameters (in addition to the stream samples) that can be defined by the user. The user populates these parameters during the application of the operator. For example, a user may

define a low pass filter operator. This operator will receive as input two parameters: the cutoff frequency and the number of taps. In addition, the operators have access to the stream meta-data (frequency, number of samples, etc) during runtime and the user can incorporate these into their code.

The user defines what type of streams this operator can handle: univariate, multivariate, text, or a combination of streams of different types. The user also defines the output stream type. Although an operator may receive several streams as input (combinatory operators), it will always output a single stream for each group of inputs.

5.4.6.3.1 JVM based operators (Java / Scala)

When a user adds a new operator it is embedded within the source code of an operator class that is inherited from the base Operator class and then compiled by the application server. All compilation errors are reported to the users. Both Java and Scala language operators are serialized and stored in the operator database. When the user executes an operator, a class is instantiated by de-serializing the operator and utilizing JVM reflection. During operator execution, each node in the cluster instantiates and instance of the operator and the stream data are passed to the operators as arguments.

5.4.6.3.2 Python based operators

The main difference between Python based operators and JVM based operators is that Python is an interpreted language in contrast with Java and Scala, which are compiled. Msgpack is an efficient binary serialization protocol that was selected to pass data between the application server (that is JVM based) and the Python operators.

When a user adds a new operator in Python, the application server tests the operator by applying it to a small array containing test data. This step is done to detect possible syntax errors, although this is by no means comprehensive as different inputs may lead to execution of different parts of the code. All execution exceptions are reported to the user. A prefix and suffix are added to the user code. These are responsible for serializing the stream data to serve as operator input, and serializing the operator output to be received by the framework. The operator code is stored in the operator database. When the user executes an operator, multiple Python

processes are executed on each node to run the operator code on the stream segments. The results are collected and can serve as input for the next iteration of operators.

5.4.6.4 Operator classes

When creating an operator, the user must define which class the operator belongs to: transformation, aggregation, combination, or selection. Each class has a well-defined behavior:

Transformation – a transformation operator receives an array of data as its input, processes the data, and outputs an array of data. The input and output arrays may either be univariate (an array of samples) or multivariate (an array of arrays of samples) and they do not have to be similar (i.e. the input can be univariate while the output will be multivariate). Transformations do not require the data to be shuffled between nodes therefore data locality is maintained. The transformation is applied to each block and there is no ordering of these applications in order to leverage parallelism.

Aggregation – an aggregation operator receives a single array of data as its input, processes the data, and outputs an array of data. The input and output arrays may either be univariate or multivariate and they do not have to be similar. Aggregations are used to combine data from several blocks into a single block therefore data are shuffled between nodes. The input array contains the data of all of the blocks of the stream, so it is in essence operating on the entire stream.

Combination – a combination operator receives multiple arrays of data as its input, processes the data, and outputs a single array of data. The input and output arrays may either be univariate or multivariate and they do not have to be similar. Combinations are used to combine data from several streams into a single output steam therefore data are shuffled between nodes. Combinations are applied at the block level – so a single instance will receive one block from each stream as input and output a single block of an output stream. All of the input blocks will have a start-time within the same range.

Selection - a selection operator receives a single array as its input, processes the data and returns a single Boolean value. The Input array may either be univariate or multivariate. Selections are used to filter out streams from a session based on a criterion. For instance, the user may decide to filter out any stream which contain values that are greater than some value x. The selector would return true for a block containing a value greater than x, and the system would remove all streams that had one of their block selectors return true. Selectors are the only type of operator that do not operate on the data within the stream, but rather on the streams themselves.

5.4.7 Bulk Data Import Service

The data importer is used to ingest data into the stream data store. The user can upload a single file or multiple files in a compressed archive. In case of the latter, in order to reduce run-time, the entire process is parallelized and runs on multiple processors. I will use the well-known Extract-Transform-Load (ETL) pattern to define the various stages of the data ingestion process.

Extract – data are extracted from various sources. A source may either be a file, or a real-time streaming source. The user provides the following information on the source:

- Source file format – the file format may be a specific binary format (such as Affectiva Q or Jawbone UP) or a CSV format. Currently two types of CSV formats are supported: timestamp value, and onset-offset time. A timestamp value CSV will contain one or more time-series defined by timestamp and value. This format is useful for data obtained from sensors (such as temperature or acceleration). The onset-offset CVS format will contain one or more time-series defined by onset and offset events. For each event a time stamp will be specified in the file. In the stream that will be created, an on-set event well be designated by a value of 1 and an offset will be designated by a value of 0. The onset-offset time format is useful for recording participant self reports in a study. For instance,

Source-id, timestamp, value	*Source-id, sleep-start, sleep-end*
CS13M001, 2014-02-12 01:55:10.100-05:00, 1.4	*CS13M001, 2014-02-12 01:55:00-05:00, 2014-02-12 08:20:00-05:00*
CS13M001, 2014-02-12 01:55:10.200-05:00, 1.25	*CS13M001, 2014-03-12 02:23:00-05:00, 2014-03-12 07:45:00-05:00*
CS13M001, 2014-02-12 01:55:10.300-05:00, 2.7	*CS13M001, 2014-04-12 01:41:00-05:00, 2014-04-12 09:15:00-05:00*

Figure 41. Time-stamp value CSV format (left) and onset-offset CSV format (right)

a participant may be asked top report sleep start and end times for each night of the duration of the study. This report can be effectively summarized using an onset-offset time CSV.

- Source-id – the source-id of the stream. If the file contains multiple streams from multiple sources, this parameter can be empty.
- Stream-id – the stream-id of the stream. If the file contains multiple streams, they will all receive the same stream-id (for example EDA, TEMP, SLEEP, etc).
- Source sample-rate (optional) – the sample rate at which the stream was recorded. In some cases, the sample rate will be included in the data file itself.
- Target sample-rate (optional) – the sample rate at which the stream should be stored in. This parameter is particularly useful when ingesting onset-offset streams, as those streams contain a single sample per event.
- Group-id (optional) – a group-id that will be registered in stream index service as part of the stream meta-data. Enables the user to search and analyze streams in a specified group.
- Study-id (optional) – a study-id that will be registered in the stream index service as part of the stream meta-data. Enables the user to search and analyze streams in a specified study.
- Timestamp format (optional) – describes the format of the timestamps within the CSV file. Appendix E contains all of the supported fields.
- Timestamp header – the CSV header of the column that will contain the timestamps in a timestamp-value CSV.
- Value header – the CSV header of the column that will contain the values in a timestamp-value CSV.
- Onset header – the CSV header of the column that will contain the onset timestamps in an onset-offset CSV.
- Offset header – the CSV header of the column that will contain the offset timestamps in an onset-offset CSV.

Transform – the data are transformed into a unified format for querying and analysis purposes. During this stage, three data structures are created:

1. Stream sample data – a list of timestamps and for each timestamp an associated value. The value may be univariate or multivariate.

2. Stream meta-data – includes stream information (such as source-id, stream-id, and sample rate) as well as various aggregation for fast statistics during querying (such as stream minimum, maximum, and average). The stream meta-data includes all of the parameters mentioned in the Stream Index Service section.
3. Stream low-resolution cache – a low-resolution (sub-sampled) version of the original stream. This version is used for rapid display as a result of user queries.

Load – the data are loaded into persistent data storage. The stream samples are loaded in to the Core Database Service, the stream meta-data are loaded into the Stream Index Service, and the subsampled stream version is loaded into the Stream Low-Resolution Cache Service.

5.4.7.1 Ingesting batch data

The user can upload a batch of data files. The files can either be uploaded separately or as part of an archive (zip) file. A file can be either in one of the supported sensor formats, or in CSV format. A single file can either contain a singe stream or multiple streams / source-ids. In case of the first, the user provides the source-id and stream-ids for that file. In case of the latter, two modes are supported:

1. The meta-data are located within the file and the user needs to configure which columns contain the stream-ids and source-ids.
2. The meta-data are part of the file-name / directory name and the user needs to provide a regular expression that enables the system to extract the meta-data from the file or directory names.

After the data are uploaded the system checks whether the data are part of an existing stream or whether the data are a new stream. If part of an existing stream, the system will make append/prepend the new data (depending on the timestamps), update the stream metadata start and end times to include the new samples, as well as update the gap metadata structure. If the data are a new stream, the system will create a meta-data structure in the stream index service. Finally the new data will be low-pass filtered and smoothed and stored in the low-resolution cache for fast retrieval.

5.4.7.2 Ingesting streaming data

Data can be uploaded to the system in real-time directly from the sensor, or from an aggregator. The stream data are uploaded using the HTTP REST interface. Each frame will contain the source-id, the stream-id, the samples and their time stamp.

5.4.8 Web User Interface Server

The web interface server is implemented in Ruby on Rails (or Rails) (Hartl, 2012) a web application framework. Rails is a Model View Controller (MVC) framework that encourages the user of web standards for data transfer, and HTML, CSS, and JavaScript for display and user-interface. The analytics engine user-interface uses Ember, an open-source MVC JavaScript application framework that enables developers to build scalable single-page web applications. The web server communicates with the application server via HTTP requests. The application server provides a well-defined RESTful interface over HTTP.

5.5 Approach Limitations

The current architecture has several limitations:

1. Constant sample rates – each stream is assumed to have a constant sample rate. Multiple sample rates per stream are not supported
2. Streams are segmented into non-overlapping blocks. This may cause limitations on operators that are trying to extract features that fall within 2 neighboring blocks. This may be mitigated by a 2 step approach:

   Step 1 - for a block of size n > k and a window of size k the features are extracted for windows k+1 to n-k

   Step 2 – The first and last windows (1 to k, and n-k+1 to n) are aggregated to one node and the remaining features are extracted from them.

It can scarcely be denied that the supreme goal of all theory is to make the irreducible basic elements as simple and as few as possible without having to surrender the adequate representation of a single datum of experience

Albert Einstein

6 User Interface Design and Implementation

6.1 Design Considerations

We wanted the user interface to reflect our design approach, which is well embodied in the quote above by Albert Einstein: provide an interface that is as simple as possible, which will provide an intuitive interaction to the user, while still providing enough flexibility to enable customization. Rogers (Rogers, 1995), claims that in order to reduce the friction of the adoption of an innovation, it should be compatible with pre-existing systems and be easy to learn and evaluate. Providing an over complex interface with many "knobs" and "buttons" may hinder user acceptance and serve as barrier to adoption. The stream search engine should provide a familiar interface that the user is already accustomed to, similar to web search interfaces, such as Google. The query syntax should have very few keywords, and enable the user to utilize natural language to retrieve data. In terms of trialability, the interface was designed so that the user could not do any "harm" to the data: the user may query and manipulate the data at will, but the original data will remain unchanged in the database. The original data are immutable as they reflect the signals as were sampled by the sensor. Those samples may contain various artifacts and the user is free to remove those artifacts in subsequent manipulations – but such artifacts may often be found to contain pertinent information, and therefore the user should not have the ability to remove them from the original data. Finally, the user should be allowed to quickly view a large part of the dataset at very low resolution, and have the ability to zoom-in to various areas of interest. This interaction contributes to the pliability of the data.

6.1.1 Logging in

The entry point to the tributary system is the login screen. The user can enter their access credentials: user-id / email and password or create a new account.

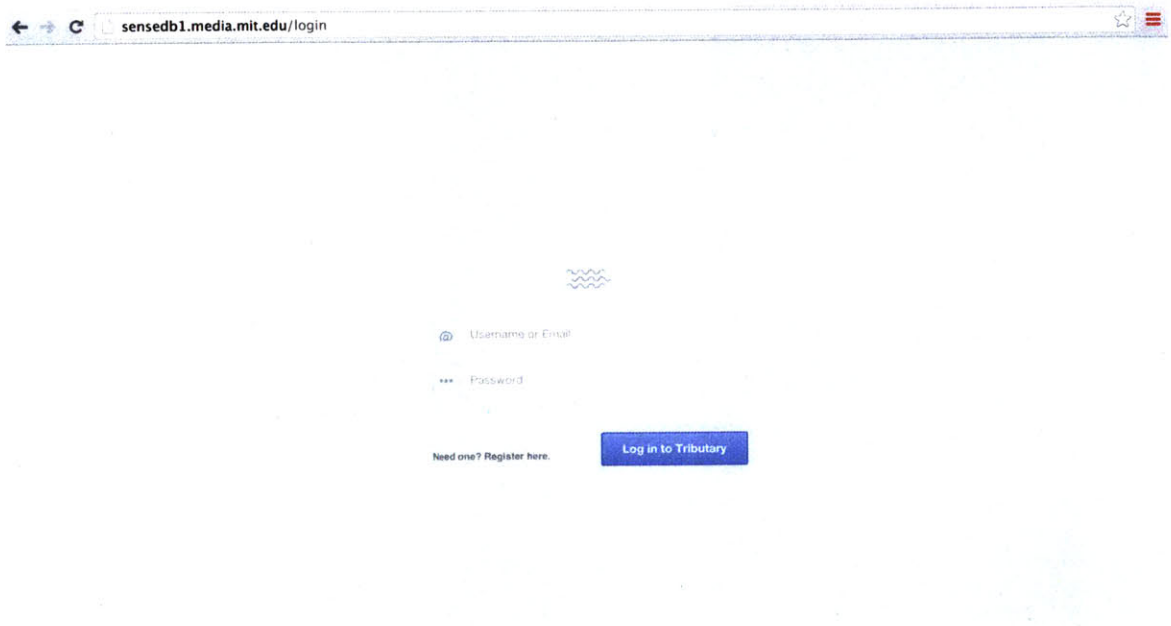


Figure 42. The Login Screen

Multiple users are able to login and access the system concurrently. Once the user is logged in, all analytics results are persisted, until actively deleted by the user. Users can logout from one location and login from a different one (for instance home and the office) and have access to their latest persisted sessions.

6.1.2 Data Exploration

After logging-in the user can explore the data by entering a search query (for a detailed explanation on the query syntax please refer to the *Stream Index Query Language* section in this chapter).

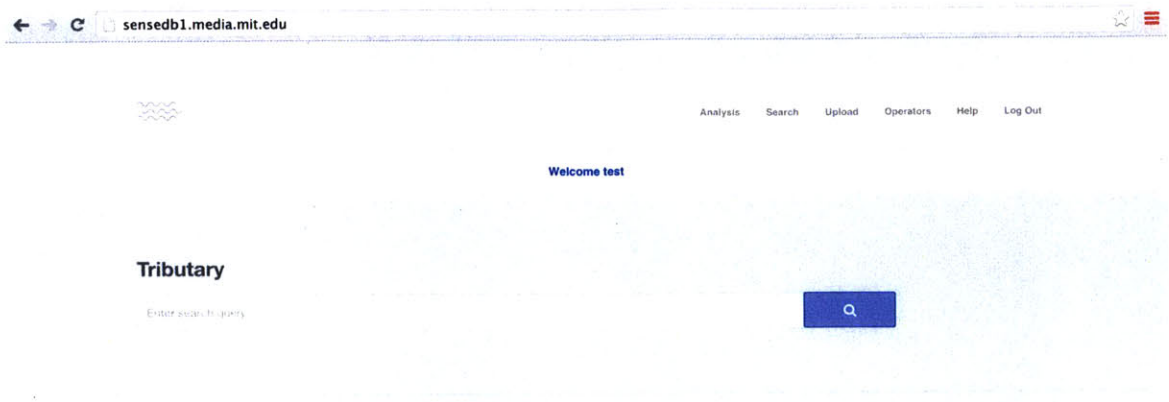


Figure 43. The stream query screen

After the user enters a search query, the stream query screen is populated with the results (as can be seen in image11). Each results page shows 10 streams, and the user can navigate between the result pages using the *'Previous Results'* and *'Next Results'* buttons. The segmentation of the results into pages was done in order to shorten the system response times, as each stream preview contains a significant amount of samples that need to be sent to the browser. At the top of the search results there is a summary that specify how many streams were returned by the query, as well the aggregate number of hours, day and samples (image 12). Each row of the search results is a tile that contains a preview of a single stream along with its meta-data. The following meta-data appears:

Max: the global maxima of the stream

Min: the global minima of the stream

Average: the arithmetic mean of the stream

Sample Rate: the sample rate in Hz at which the stream was sampled.

Owner: the owner of the stream is the user that imported that stream into the system.

Days: the number of days that the stream was recorded in.

Completeness score: $\frac{actual\ number\ of\ samples}{expected\ number\ of\ samples}$

This is a measure for the completeness of the data for that stream.
The expected number of samples is calculated as follows:

$$expected\ sample\ num = \frac{(end\ date\ milliseconds - start\ data\ milliseconds)}{1000} sample\ rate\ Hz$$

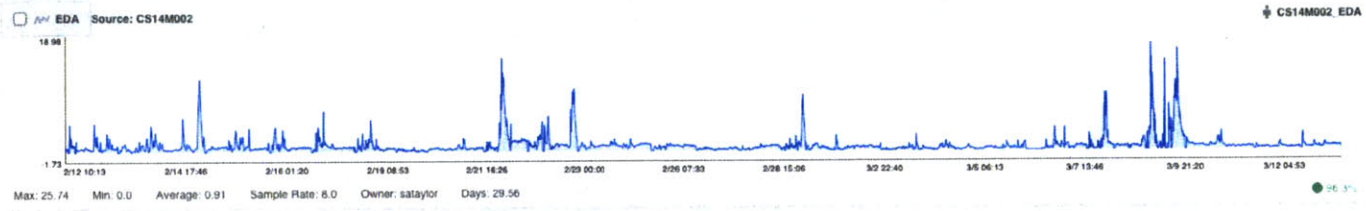


Figure 44. Result tile containing stream preview and meta-data

It is possible to zoom in to a specific period of time by clicking the left mouse button and dragging the mouse pointer. Double clicking will zoom out.

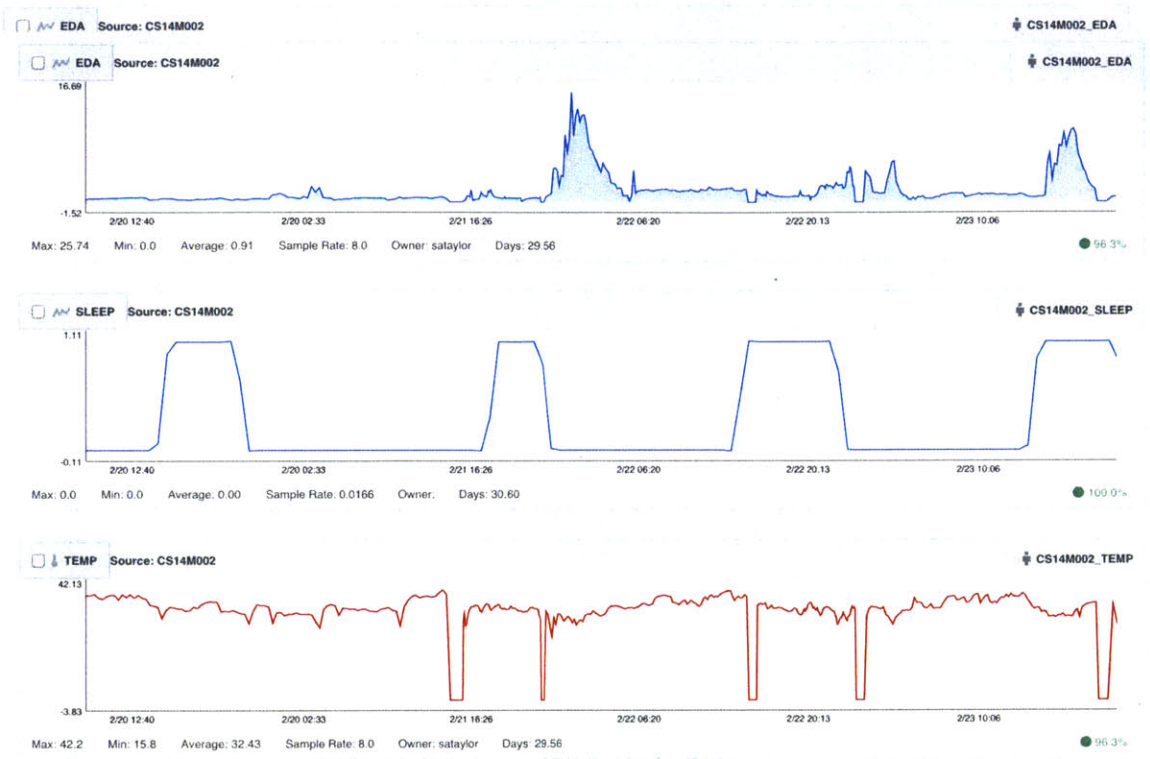


Figure 45. Multiple stream zoom-in

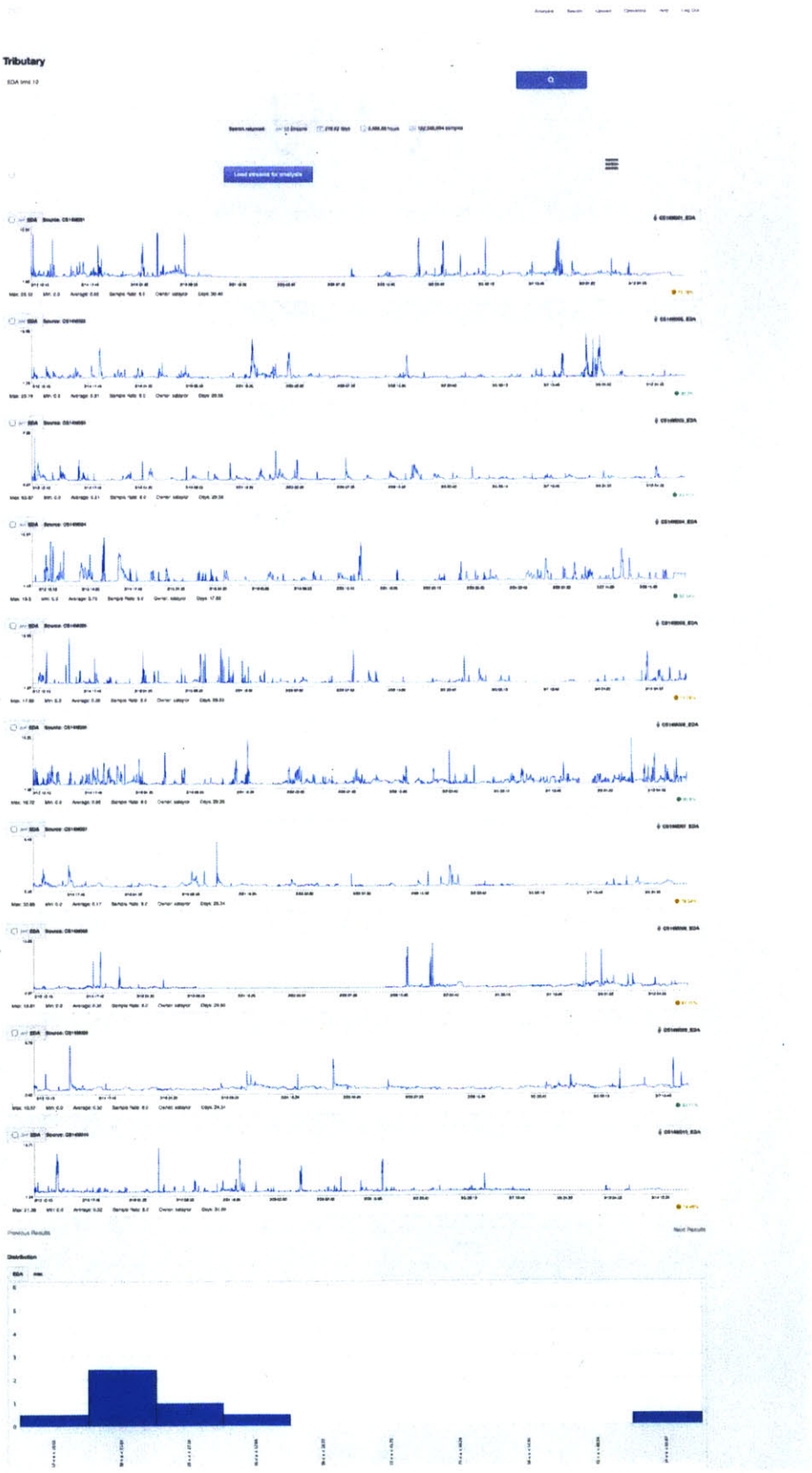


Figure 47. Stream Query Results

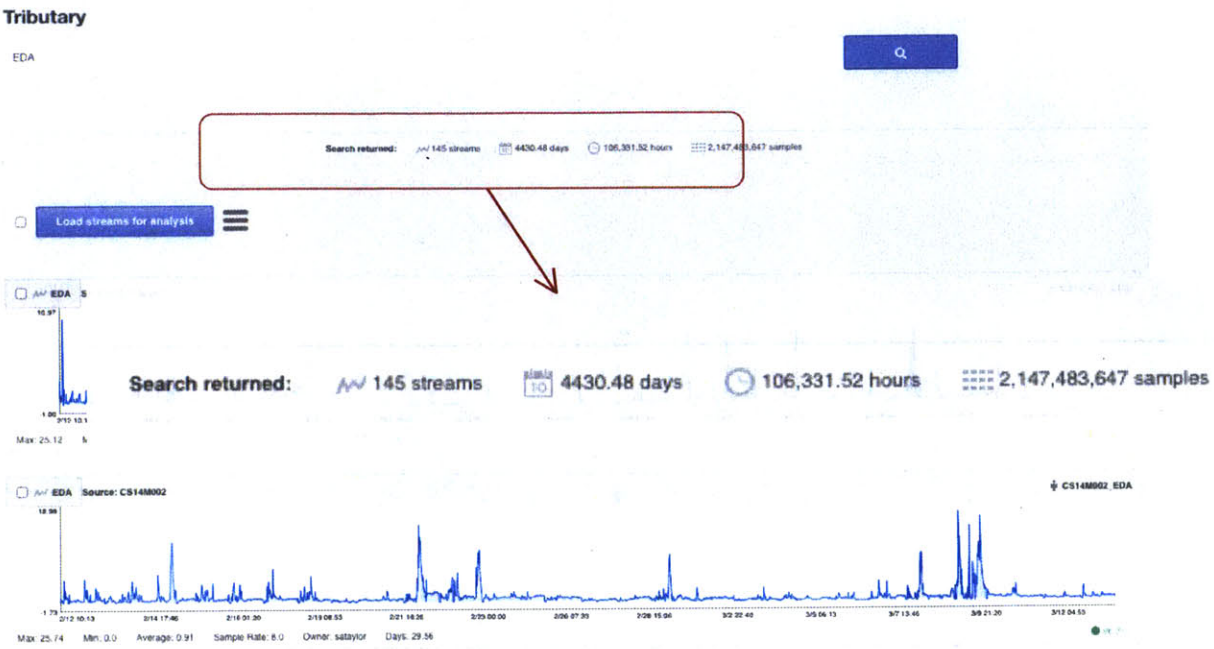


Figure 48. Search results summary

When zooming into a single stream, all the other streams in the result page will be zoomed to the length of time. This is specifically useful when needing to compare between several streams at a higher resolution (such as hours).

Users can select streams for analysis by selecting a checkbox at the top right of the stream preview tile. Once one or more streams have been selected, the user can click the "*load streams for analysis*" button. The button will display the number of streams that are going to be loaded for analysis. If the user does not select any streams and clicks the "load streams for analysis" button, all of the streams that were returned by the query will be loaded for analysis (including those in other pages). The user can also select in what units the streams will be partitioned across the cluster: hours, days, or weeks (figure 15). If an operator needs access to data samples of a specific period consecutively, the user would select the partition accordingly.

Figure 49. Stream partition configuration

6.1.3 Data Analysis

Once the user has explored the dataset and selected streams for analysis, the analysis tab will be populated by the user analysis sessions. Each session will contain the data for a specific sensor type. For instance, if a user has loaded temperature and acceleration data all temperature data will be loaded into one session, and the acceleration data will be loaded into another. The user can choose which session to view by selecting the session number under the analytics tab. Each session will initially display a root tile. This tile represents all of the data that was loaded into the session (Figure 16). Within the tile, a 20-minute random preview of 2 streams of raw data is displayed. This enables the user to establish if there are constant artifacts in the signal. The 2 streams that are displayed are always the first two streams as ordered by their source-id. The user can select an additional stream preview to view by selecting it from a dropdown list that contains all of the streams loaded into that session (Figure 17).

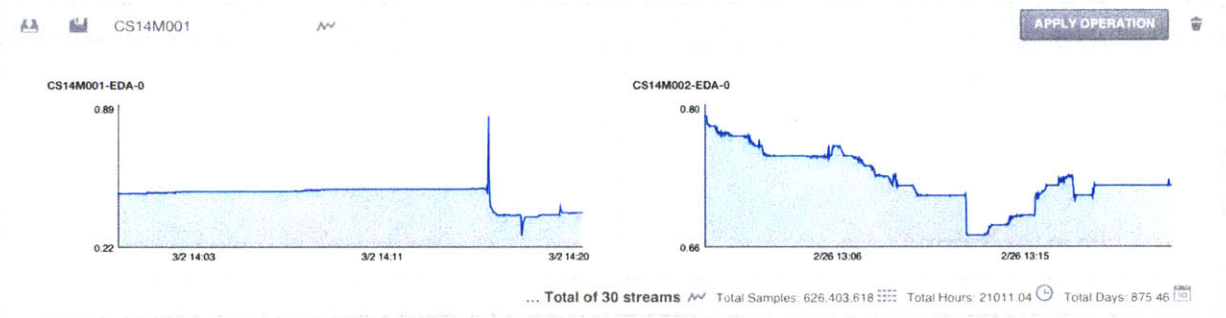


Figure 50. Analytics session tile

The tile also shows how many streams are currently loaded into the session, the total amount of samples, hours and days.

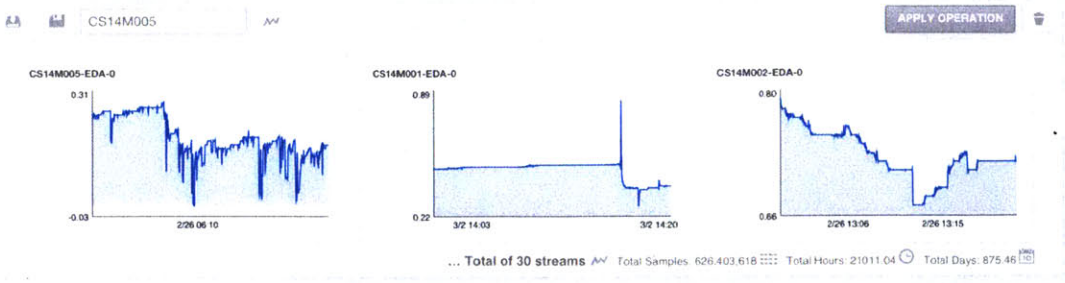


Figure 51. Analytics tile containing additional stream selected by user

The user can view a down-sampled versions of all the streams within a session node by clicking the "*total of n streams*" link at the bottom of the tile. This operation will execute a sampling and low pass filter on each of the streams segments in the cluster and then retrieve the results for display.

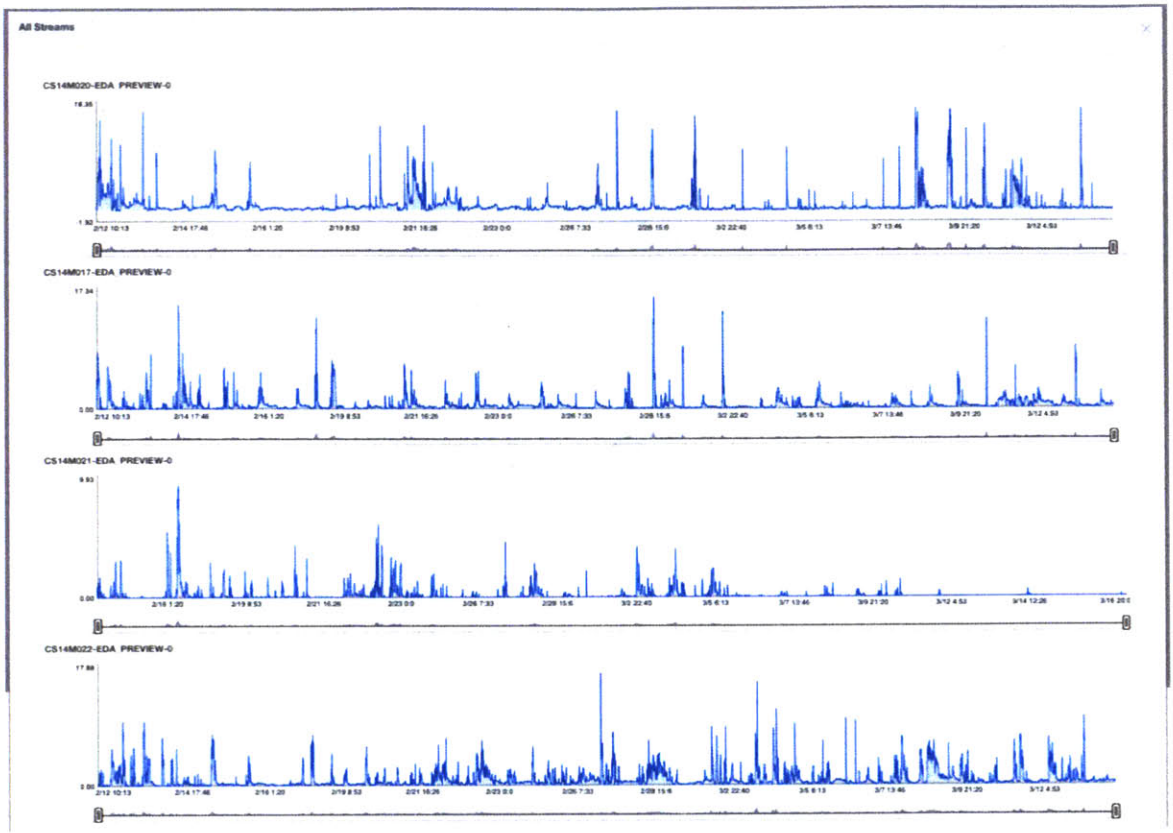


Figure 52. Preview view of all the stream loaded into a session node

Users can apply operators to each tile (computation node) by clicking the '*Apply Operation*' button on that tile. After that, a dialog containing all of the operators the user has access to will appear categorized by operator class (figure 19).

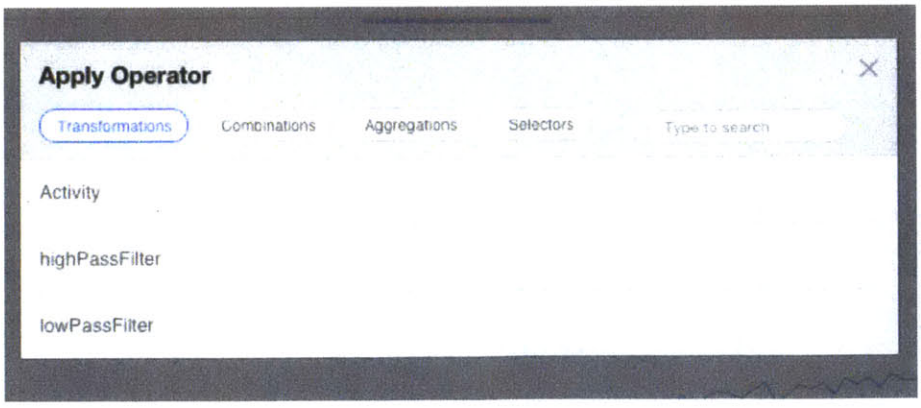


Figure 53. Apply Operator Dialog

After selecting an operator, the user can enter various parameters that are required by that operator. For instance, in figure 20 the user is required to enter the cutoff frequency and the number of taps.

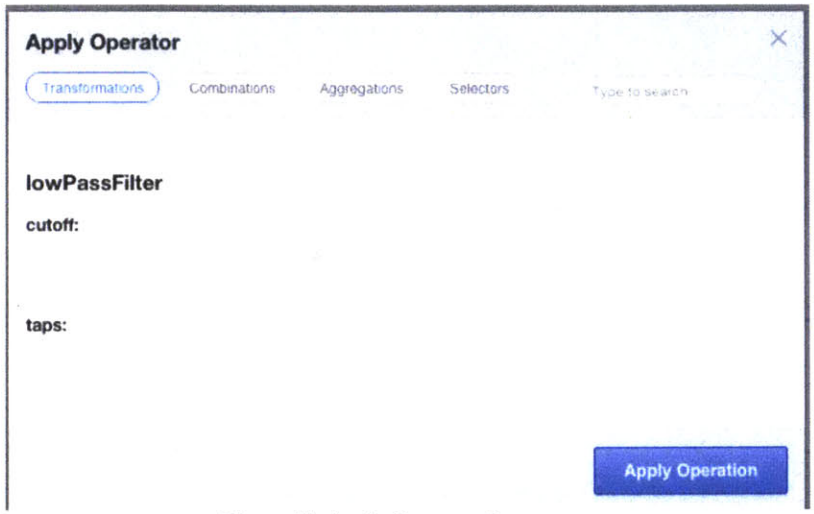


Figure 54. Apply Operator Parameters

After entering the parameters, the user can apply the operation by clicking the "Apply Operation" button. The system will present a progress bar that displays the estimated progress of the operator (figure 21). Operator execution time varies greatly and depends on several factors: the operator computation complexity, the size of the data set, and the number of cores in the cluster. In some cases, the operator may complete within a few hours. Therefore, the system also sends an email to the user notifying them of the completion of the operator and its completion time.

Figure 55. Operator progress bar

When the operator has completed, the result of the operator is displayed below the tile that the operator was applied to. In the image below (figure 22) the user has first applied a low pass filter (the result is displayed in the 2nd tile from the top) and then a maximum operator (3rd tile from the top). The user can re-apply the same operator with different parameters or apply a new operator to a tile in order to compare between the results. In figure 21 the user has applied a low pass filter 3 times. Each time a different cutoff was used. The user can compare between them and select the desired result, before continuing with subsequent computations.

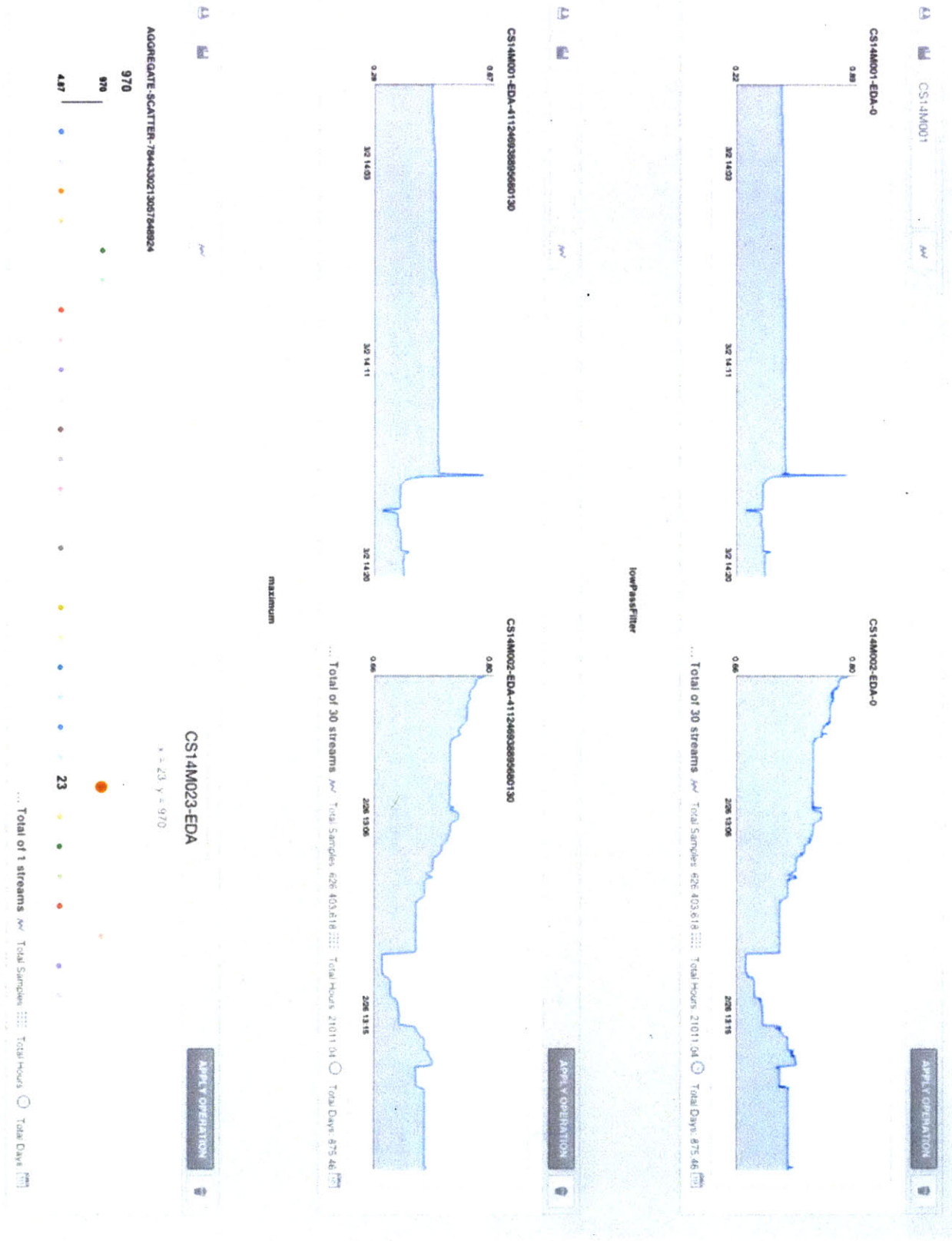


Figure 56. Analytics View – 30 streams are loaded into the analytics engine. The top tile shows a 20 minute preview of 2 streams. A low pass filter is applied to all 30 streams. The preview of the result is displayed in the 2nd tile from the top. An operator calculating the global maxima of each stream is applied. The result is show in the last tile. The cursor is hovering above one of the streams in the scatter plot, and the global maxima for that stream is displayed

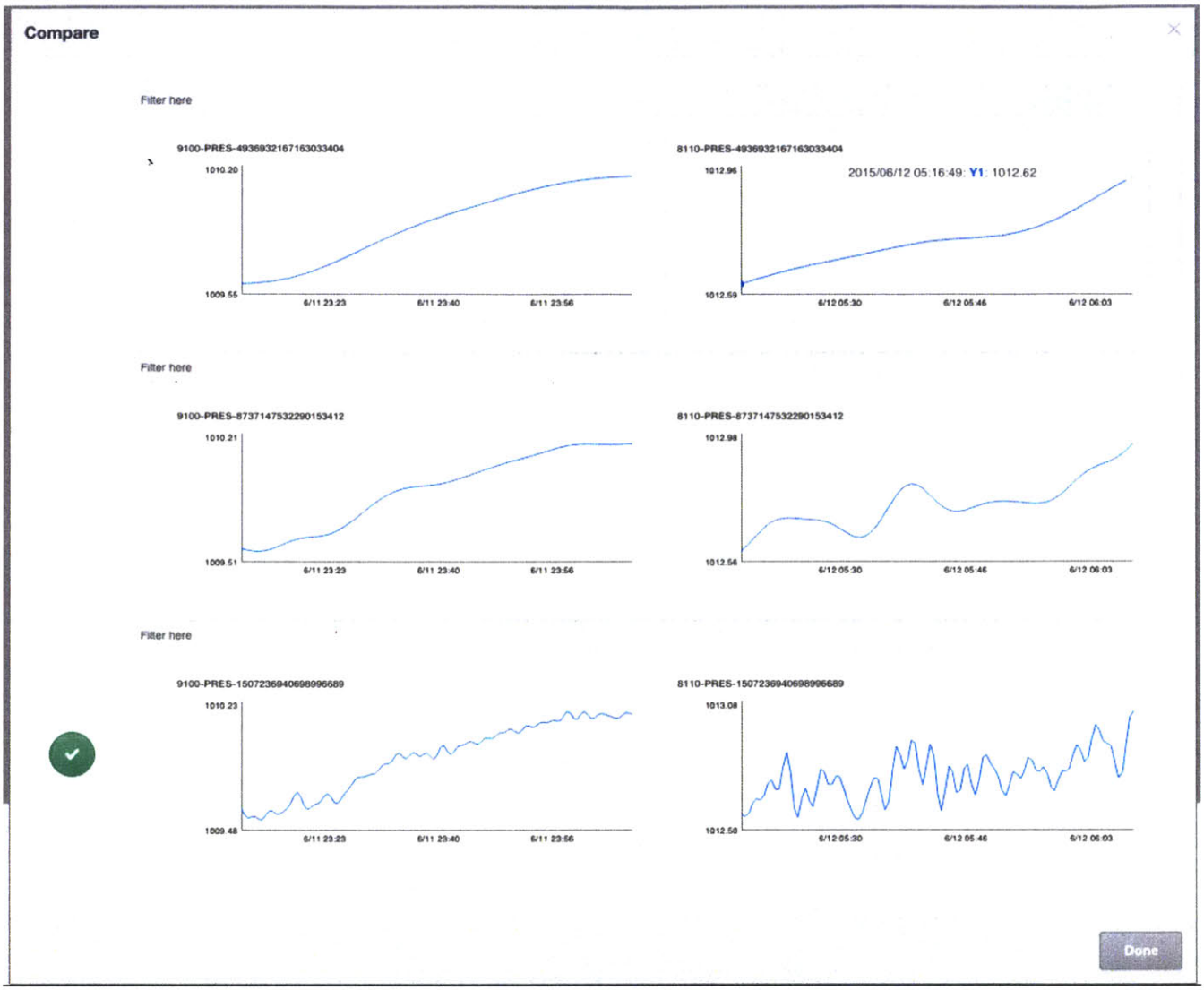


Figure 57. Comparing 3 operator results

At any point, the user can go back to a node that has multiple results (siblings) and select a result to continue the computation with. The other non-selected child nodes are swapped out. A tile that contains multiple results is represented by a stack of tiles. The top left field displays the number of siblings (figure 43).

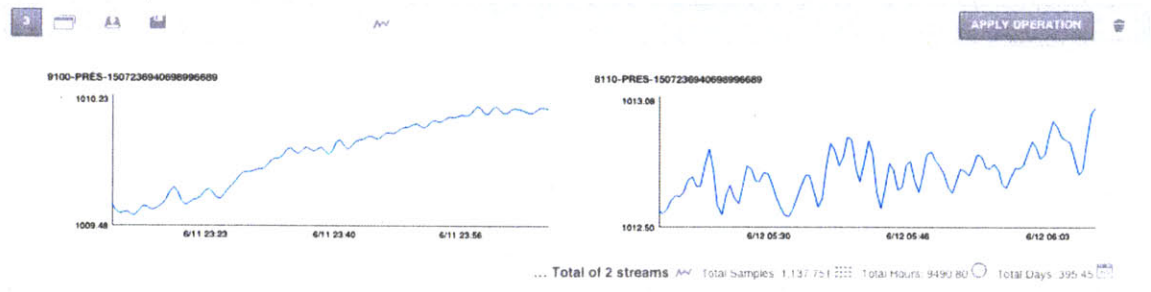


Figure 58. Stacked tile with 3 results

6.1.4 Writing Operators

Users can write their own operators or edit existing operators by clicking on the operator tab. A list of existing operators will be displayed and the user can either select one, or click the '*Create New Operator*' button. The operator editor enables the user to edit/create an operator as well as configure additional operator related parameters (figure 25).

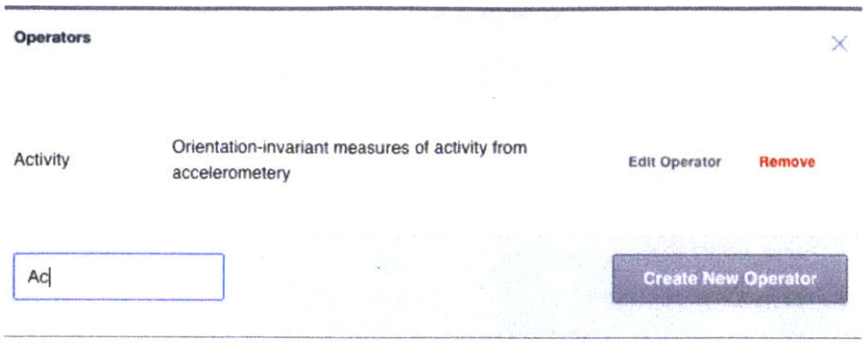


Figure 59. Create / Edit operator list

The user writes the operator code as well as the following parameters:

Name – the operator name. This name will be displayed in the operator list when a user clicks 'apply operator' within the analytics view.

Language – the programming language. The user can select Python, Java or Scala.

Description – a detailed description of the operator that can be used by other users to understand what the operator does.

Operator type – the class of the operator: Transformation, Aggregation, Combination, or Selection.

Arguments – the names and the types of input arguments in the following format: *name1 type1, name2 type2,..., nameN typeN.* The supported types are: *Double*, *MultiDouble*, Model, *Number* and *Text*. *Double* and *MultiDouble* designate univariate and multivariate stream types respectively. The *Model* type is a Python scikitlearn machine-learning model that the user can upload via the upload interface. *Number* and *Text* are designated for operator parameters (such as cutoff frequency for instance). Once the user has defined the arguments and their types they can refer to them in the operator code.

Plots - the user can select the following visualizations for the operator output:

Line – a curve plot. This visualization is useful for transformations. It is the default representation of a time series in the system. The user can hover their mouse over the plot to get detailed times and values.

Scatter – a scatter plot. This visualization is often used for aggregations and provides and easy measure to compare between the aggregated stream values.

Bar – a bar plot.

Histogram – represents the distribution of the data in the result.

Output type – the output type of the operator: *Double* or *MultiDouble*. Specifies whether the output is univariate or multivariate.

Once all of the fields have been field, the user can save the operator by clicking the *'update operator'* button. The system will check the correctness of the code by executing the operator on a small dataset of sample data. This serves only as a basic sanity test for the operator.

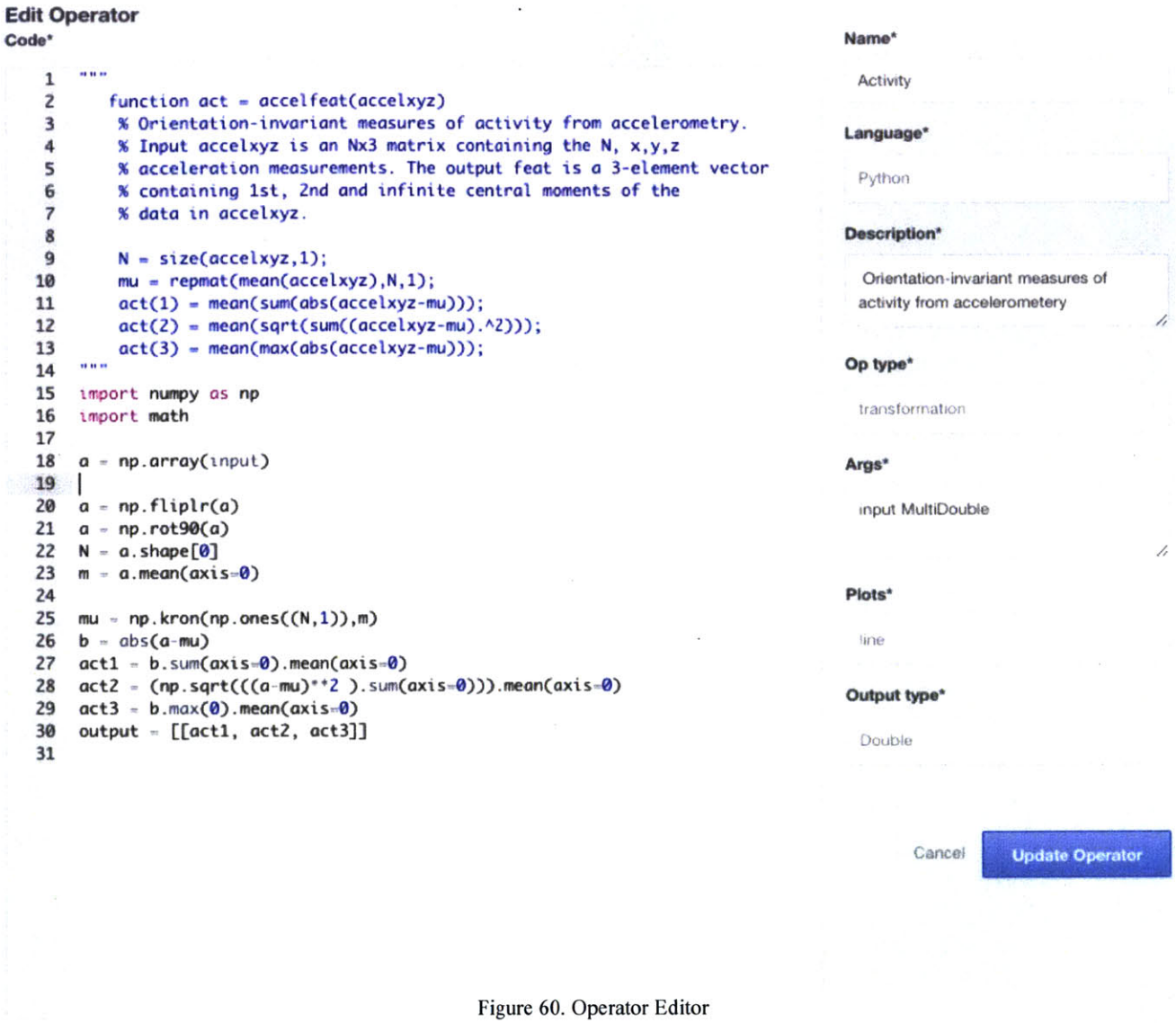


Figure 60. Operator Editor

6.1.5 Data Upload Screen

In order to batch upload the user can log in to the web interface and select the upload tab (figure 28). The user fills in a form with the following fields:

Sensor Data File – this is the filename on the local computer to upload to the system

Source ID – the source id of the file if this file only includes a single source-id. Otherwise this should be left empty.

Group ID – group id if these streams are to be assigned to a group, otherwise should be empty.

Study ID – study id if these streams are to be assigned to a study, otherwise should be empty

Source ID Header – if the file contains multiple sources, this field designates the header under which the source-id will be located

Timestamp Format – a Joda based timestamp format string ("DateTimeFormat (Joda t ime 2.2 API)," n.d.).

File Format – the user can select from a list of supported formats. In case the specific sensor's format is not support, the user can use the CSV format.

Target Sample Rate – a target sample rate

Source Sample Rate – the original streams sample rate

Other Optional Parameters – various additional parameters relevant to specific formats such as CSV.

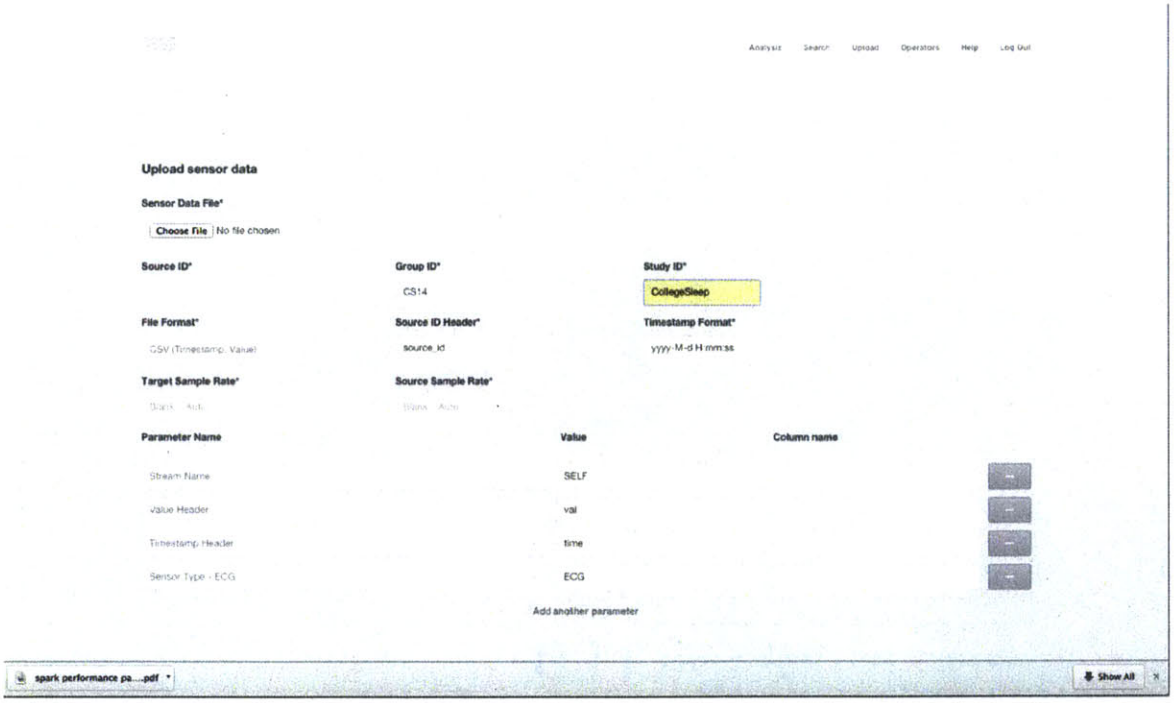
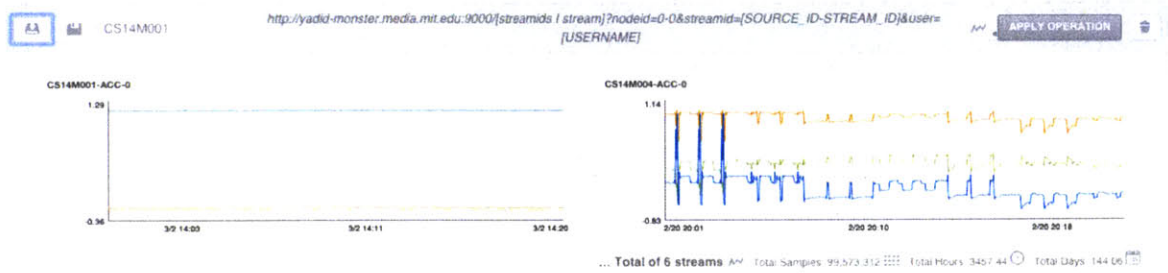


Figure 61. The data upload interface

6.1.6 Exporting data

The user can export data from each of the computation nodes in order to visualize the data outside. In order to access the data, the user can click the export data on a button located at the top left of the computation node (figure 29). A URL will be shown which the user can copy



and paste into an external tool (such as Matlab or Python) that support HTTP connections. The data are exported in JSON format, which is supported by many programming languages.

Figure 62. Export button and URL

Fear cannot be banished, but it can be calm and without panic; it can be mitigated by reason and evaluation.

Vannevar Bush

7 System Evaluation

The system evaluation chapter comprises two parts. The first part is a quantitative characterization of the system performance in a variety of scenarios. The second part contains the results of a user study: researchers used the system to analyze data that they had collected in their own studies. For the evaluation, 4 clusters were set up. The first cluster was located within the MIT Media Lab data center, and contained 7 machines and a total of 45 cores and 70GB of RAM allocated to the data. The second, third and forth clusters where located on the Microsoft Azure cloud platform. They contained 12 VMs with 88 cores, 5 machines with 40 cores, and 45 machines with 180 cores. These clusters had 66GB, 30G, and 450GB RAM allocated for data respectively. All in all throughout this study a total of 69 VMs were installed and provisioned.

7.1 System Performance Characterization

System performance was evaluated on 4 dataset sizes: 100 streams, 50 streams, 25 and 12 streams. Each stream was composed of 30 days (1 month) of a physiological signal recording by a wearable sensor at a rate of 8Hz. As the participant did not wear the sensor for the full 30-day period (due to the need to charge it, or shower) the recording contained a net of 22 days of data or 528 hours, which are 15.2 Million samples. Each sample was stored as an 8-byte double precision value. The size of each raw stream was 231.8MB not including its metadata that took up an additional 120KB. The table below shows a summary of the 4 datasets.

Name	Sample Rate	Signal type	Hours	Number of streams	Number of samples	Raw size
Dataset 1	8Hz	EDA	52800	100	1.52 Billion	22.6 GB
Dataset 2	8Hz	EDA	26400	50	760 Million	11.3 GB
Dataset 3	8Hz	EDA	13200	25	380 Million	5.64 GB
Dataset 4	8Hz	EDA	6600	12	190 Million	2.78 GB

Table 1. Dataset Summary

Each data set was evaluated on 4 different cluster sizes: 46 nodes, 23 nodes, 12 nodes and 6 nodes. The nodes were virtual machines hosted on the Microsoft Azure platform. Each VM was running Ubuntu 14.04.2 LTS (GNU/Linux 3.16.0-37-generic x86_64) and had the following packages installed:

-Oracle Java 1.7.0_80-b15
-Scala 2.10
-Python 2.7.6
-Numpy 1.8.2
-msgpack 0.4.6
-Apache Spark 1.4.1
-Apache Cassandra 2.1.5

Each slave node had 4 Intel(R) Xeon(R) CPU E5-2673 v3 processors with 30720 KB cache running at a clock speed of 2.4GHz. Each node had 28GB of RAM, 285GB hard drive running the OS and software, and a 1TB hard drive storing the data.

The master node had 8 AMD Opteron 4171 HE processors running at a clock speed of 2.9GHz with 512KB cache. It had 56GB of RAM and a 605GB hard drive running the Operating System and software.

The table below shows a summary of the 4 cluster configurations:

Name	Number of slave nodes	Total number of cores	Total RAM allocated to data
Cluster-1	46	184	276 GB
Cluster-2	24	96	144 GB
Cluster-3	12	48	72 GB
Cluster-4	6	12	36 GB

Table 2. Cluster configuration summary

For each dataset all classes of operations were profiled: loading, transformation, aggregation, combination, and selection. For each operator a representative example was created. Each of the scenarios was executed across all the cluster sizes. The details of each tested operation are as follows:

Loading – all of the streams were loaded into the cluster RAM using the default partitioning of 1-hour blocks.

Transformation – a low pass FIR filter with a cutoff of 0.0001 and 64 taps was applied to each stream.

Aggregation – the maximum sample of each stream was calculated

Combination – Each stream was loaded twice (under different stream id's) and portioned across the cluster. A correlation was calculated between each pair of streams.

Selection – streams that contained a sample greater than X were discarded. An arbitrary value of X=15.0 was selected.

The rational behind profiling the various operator classes is that each of them has different computation and IO behaviors; Loading is both IO and compute intensive. It requires each node to first locate the selected streams and their time range (if one was selected), and read

them from disk, which is IO intensive. Next, each stream is segmented into blocks (dependent on user configuration) and partitioned across the RAM of the entire cluster. The memory allocation for this process is computation intensive.

Transformations are computation intensive if the dataset fits completely into the RAM of the cluster. If the dataset is larger than the available RAM, the excess data will be spilled to disk, which turn the transformation into an IO intensive operation as well.

Aggregations are mostly compute intensive; first each block is aggregated locally (compute intensive) and then all the blocks of a stream are aggregated into a single node (IO) and finally an aggregation is computed on those values (compute intensive).

Combinations are IO and compute intensive; pairs of blocks are collocated on the same node, and then combined.

Selections are compute intensive; an operation is applied to each block, and depending on the result, the stream may be removed from the node.

All test scenarios were implemented using a Scala script, which executed the relevant Tributary API functions. Each class of operator was executed 5 times in each of the cluster size scenarios and an average and standard deviation was calculated. The block diagram below depicts the general test framework:

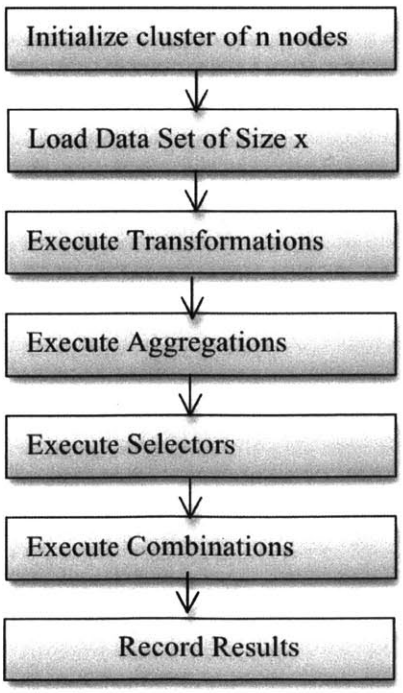


Figure 63. System Test Framework Operations

7.1.1 Data Loading

In this test, the core database service resided on 46 machines. A single key space was defined on Cassandra, which served as the stream data-store. The key space had a replication factor of 3 using the SimpleStrategy replication strategy: single datacenter in which the first replica is determined by the partitioner, while additional replicas are placed on the next nodes clockwise in the ring without considering rack or datacenter location. Cassandra's JVM heap size was configured to be 10G and the new generations maximum size was configured to be 2.4G. Concurrent read and writes were configured at 32.

In each run, the analytics engine was configured to utilize a different number of nodes for computation. Below is a chart illustrating the loading times for each dataset on the various cluster configurations (figure 2):

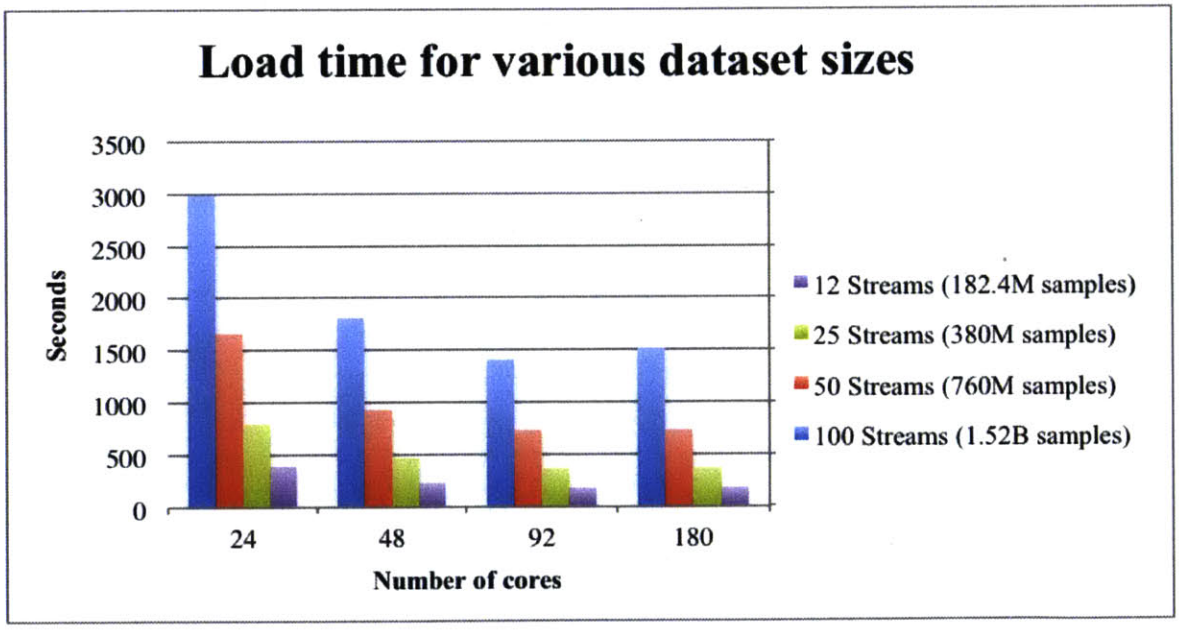


Figure 64. Stream loading times for various dataset sizes and cluster sizes

The results show that while increasing the analytics engine cluster size, yields an improvement in performance, from a certain point, increasing the cluster size does not shorten the loading times of the datasets (for all sizes). This can be explained by the fact that when utilizing a small cluster for the analytics engine, the number of nodes reading data concurrently will affect the throughput. The more nodes read concurrently the higher the throughput will be. But as the number of concurrent nodes reading grows, the data store becomes the bottleneck, as

it can only support a certain number of concurrent reads effectively. In order to improve load time performance, it would be necessary to add additional machines to the data store.

7.1.2 Operator Execution

Each operator was executed 5 times on each loaded data set. This test was repeated for the 4 cluster size configuration. The first plot (figure 3) shows the results of the transformation test across all cluster sizes and dataset sizes.

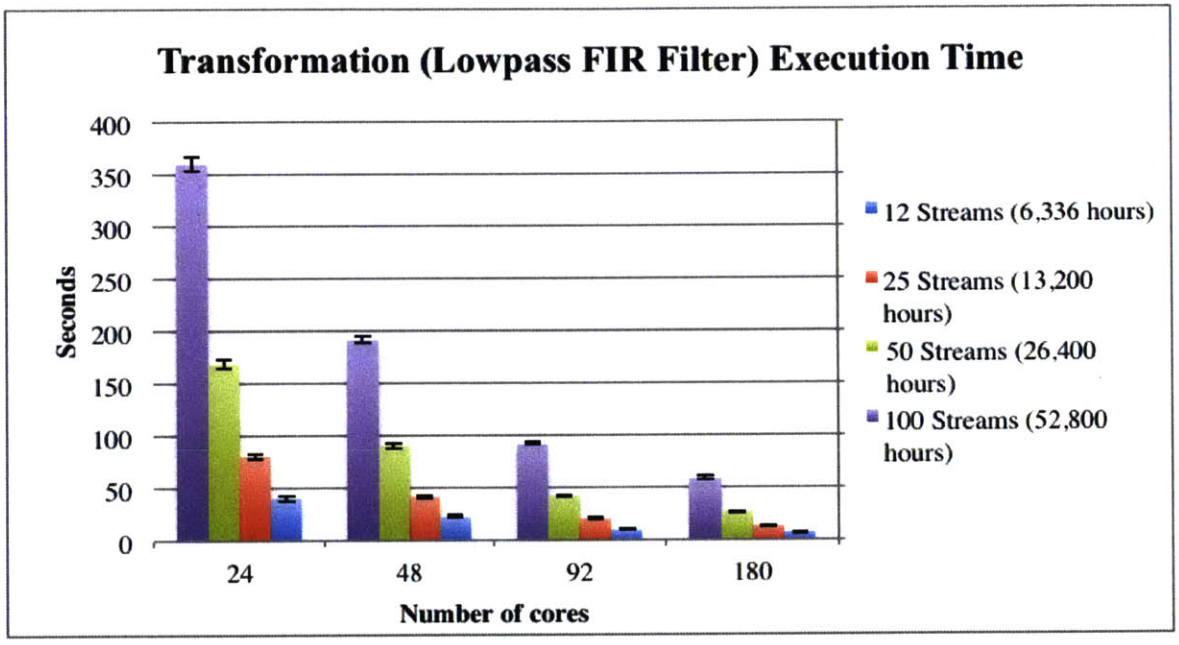


Figure 65. Transformation execution times across various dataset sizes and cluster sizes

The results show that as cluster size increases execution time for transformations is reduced by roughly the same factor. This holds true across all dataset sizes. The next plot (figure 4) shows the execution time of all operator classes across all cluster sizes for 100 streams. The results show that there is reduction in execution speed as cluster size increases across all operator classes.

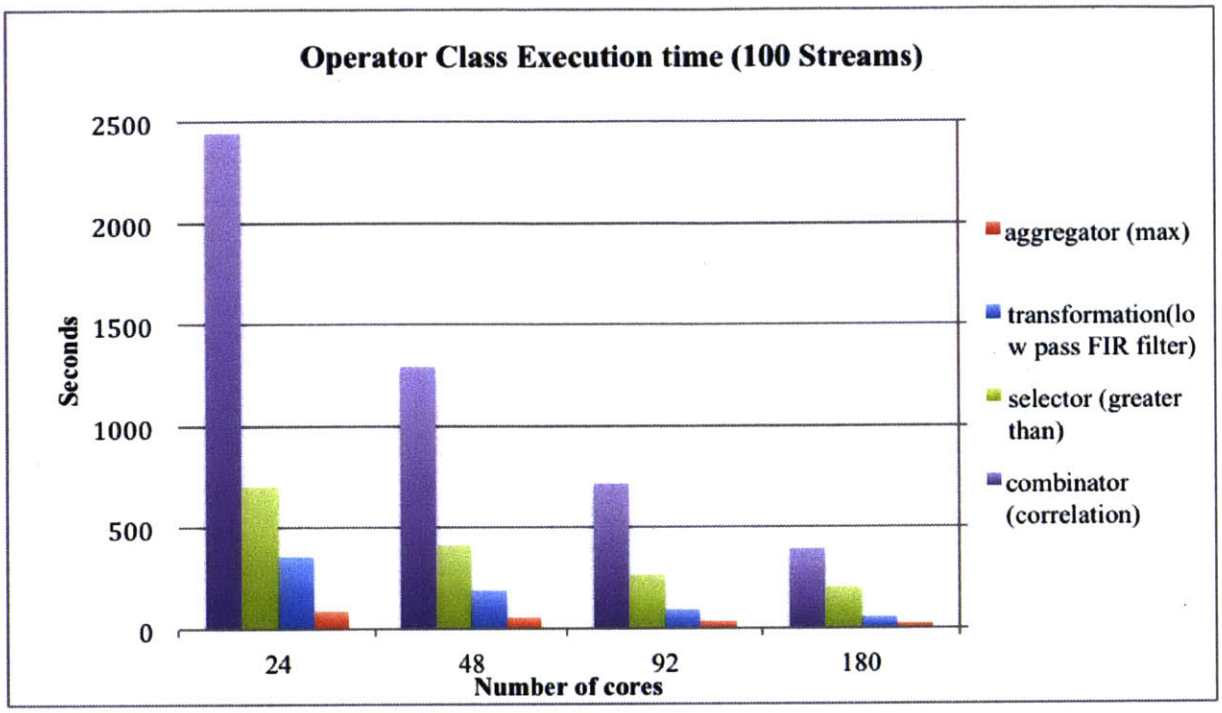


Figure 66. Operator class execution time for 100 streams across different cluster sizes

If we use 24 cores as our base case and calculate the processing speed improvement in percent, the following picture emerges (figure 5):

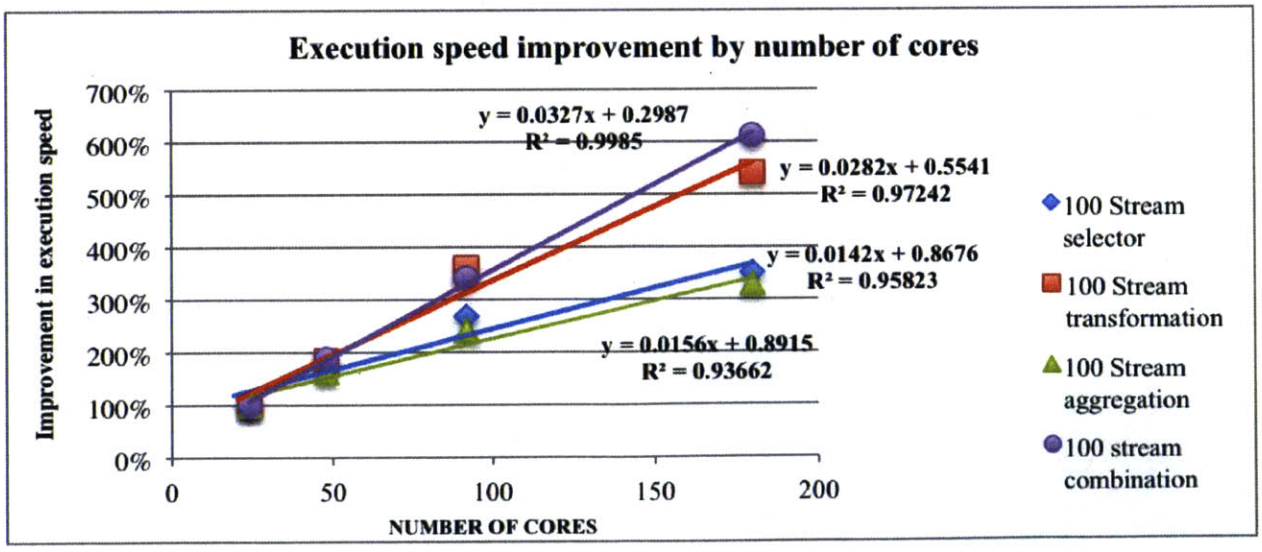


Figure 67. Execution speed improvement by number of cores

We also wanted to compare the performance of the system to that of a laptop similar to the one used by many researchers for running data analysis. Our test laptop was an Apple

MacBook pro with a 2.6 GHz Intel Core i5, 16GB 1600 Mhz DDR3 and running OS 10.9.4. The Laptop had Python 2.75 with Numpy 1.6.2 and Scipy 0.11.0 installed.

Our Python test script generated 4 data sets similar in size to those we tested on the system. Each data set contained random data. We tested a low-pass FIR filter with 64 taps and 0.001 cutoff, similar to the one we executed in our transformations.

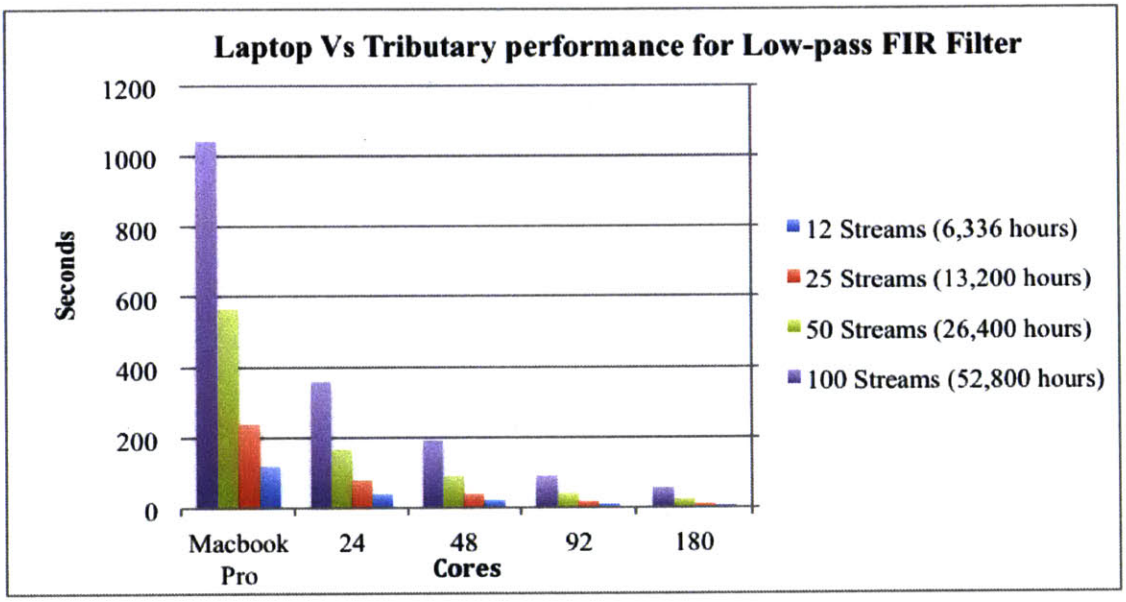


Figure 68. Lowpass FIR Filter execution time: Laptop vs Tributary comparison

The test scenario executed on the laptop and the one executed on the distributed system differ in several important aspects. The laptop test scenario only included the stream raw data (without any timestamps) whereas in the distributed system timestamps were used in addition to the data. This was done because in the case of analysis on a single monolithic machine, the samples are stored consecutively in an array, which makes maintaining the timestamps unnecessary as they can be calculated on the fly when required. As a result the distributed system had to handle the metadata associated with these streams, whereas the laptop test did not. Even with this additional load, the distributed system outperformed the laptop by an order of magnitude.

Finally we wanted to characterize the system overhead associated with running the computation on a distributed platform such as Spark. We ran a transformation on the largest

dataset (100 streams), utilizing the largest cluster size (180 cores). The results are shown in figure 7. In Spark, a task is the smallest unit of work that is run on a single core. Overhead includes the result serialization time, garbage collection time, task deserialization time, and scheduler delay.

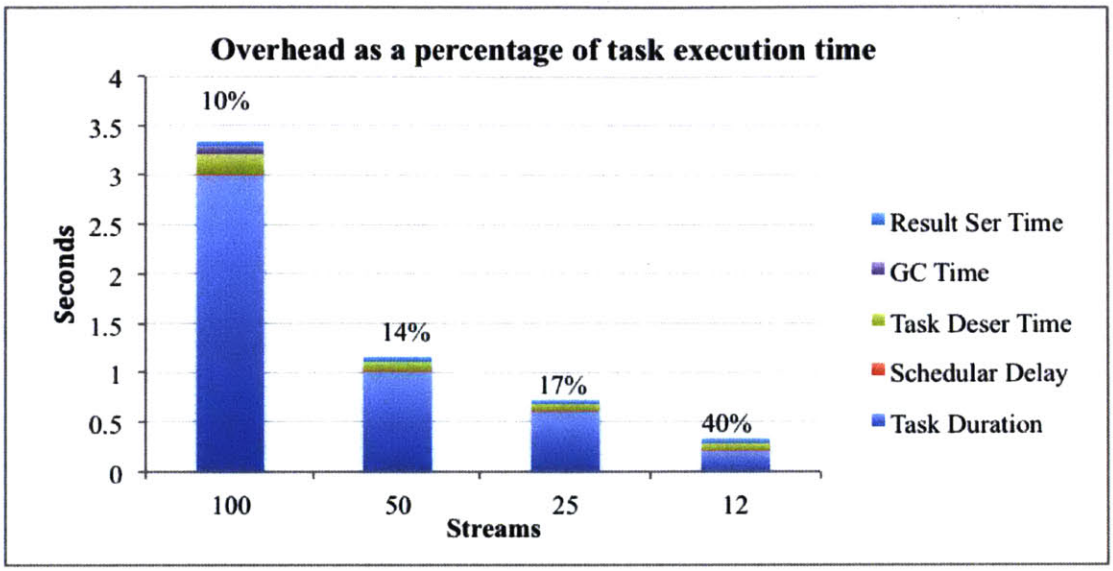


Figure 69. Average overheads as a percentage of task execution time

The results show that on the smallest measured dataset, nearly 40% of the task execution time is overhead. A possible solution would be to configure the number of tasks based on the size of the loaded dataset. In that case, a smaller number of tasks would mean that more computation would be allocated to each task, and as a result the size of the overhead would be less significant.

7.2 User Studies

The system was evaluated in 2 qualitative user studies. The first study focused on analyzing data obtained from physiological sensors, mobile phone data, and user-self report. The second study focused on analyzing data from environmental sensors. There were two researchers who were analyzing the first data set, and one researcher who was analyzing the second one. The researchers did not partake in the user experience design study. As a result, we could test the correctness of our user population assumptions that were the basis of our user personas.

In each of the studies, the researchers had collected significant amounts of data that would require many days of computation for analysis purposes, if they were to use their own traditional tools. Prior to using the new system, the researchers were asked to fill in a survey (that can be found in Appendix A) about their current analytics practices. After that they were also interviewed in order to gain additional information that may have not been disclosed in the survey. Next, the researchers uploaded their data to the system, and after a period that ranged between several weeks to several months of interaction, were asked to fill out 2 surveys. The first survey contained a standard System Usability Survey (SUS) that can be found in Appendix B. The second survey that can be found in Appendix C contained questions regarding the user's experience while using the system.

7.2.1 Study 1 - SNAPSHOT - (Massachusetts Institute of Technology, MIT Media Lab, Brigham and Women's Hospital)

Sleep is critical to a wide range of biological functions; inadequate sleep results in impaired cognitive performance and mood, and adverse health outcomes including obesity, diabetes, and cardiovascular disease. Recent studies have shown that healthy and unhealthy sleep behaviors can be transmitted by social interactions between individuals within social networks. This study investigates how social connectivity and light exposure influence sleep patterns and their health and performance by using data collected from socially connected MIT/Harvard

undergraduates with wearable sensors and mobile phones. The study will include 300 participants for a period of 5 years.

The sensors that were used in the study were as follows:

Sensors: Electrodermal Activity (EDA), Actigraph (Accelerometer + Light sensor), Smartphone data and sensors

Estimated dataset size: 100s of Gigabytes to several Terabytes

7.2.1.1 The Dataset

Data from two cohorts were uploaded to the system. The first cohort contained 99 participants, and the second one contained 46 participants. Each participant was required to wear a wearable wrist sensor for 30 days. The sensor was recording data at a sample rate of 8Hz. In addition, the participants were required to answer surveys using a mobile phone application. A summary of the dataset that was uploaded to the system can be found in the table below.

Name	Sample Rate (Hz)	Number of Streams	Total Number of Days	Total number of samples	Total Raw Data Size	Total Disk Size (+ DB overhead)
EDA	8	145	4430.48	2,147,483,647	16 GB	32 GB
Temperature	8	145	4430.48	2,147,483,647	16 GB	32 GB
Acceleration	8	145	4430.48	2,147,483,647	48 GB	64 GB
Sleep Self Report	0.016	144	4324.08	6,201,768	47 MB	94 MB
Totals	NA	579	17,615.52	6,448,652,709	80.05 GB	128.1 GB

Table 3. The College Sleep dataset summary

7.2.1.2 Prior methodology used to analyze the dataset

At the end of each cohort, data were downloaded from each sensor using a USB cable. The sensor data were stored on a file server using the following hierarchy:

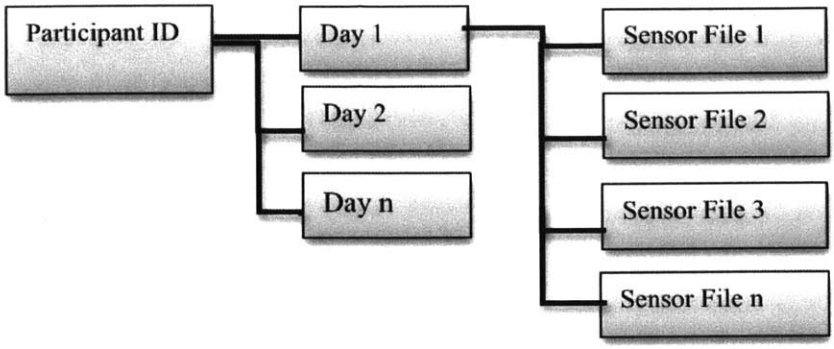


Figure 70. SNAPSHOT storage file hierarchy

Each day folder contained an average of 2 files, resulting in a total of 8700 files for both cohorts. In order to analyze the data researchers wrote mostly Python and Matlab scripts that would calculate features from the individual files. They would execute these scripts on their laptop or desktop machine. The results were managed in excel files.

7.2.2 Study 2 – Tidmarsh Living Observatory (Massachusetts Institute of Technology, MIT Media Lab)

As part of the Living Observatory initiative, researchers at the MIT Media Lab's Responsive Environments Group are developing sensor networks that document ecological processes and allow people to experience the data at different spatial and temporal scales. Small, distributed sensor devices capture climate and other environmental data. Each node contains the following sensors: temperature, humidity, barometric pressure, ambient light, and vibration/motion. In addition, it can optionally contain an audio codec DSP and expansion for additional analog and digital sensor channels. The sensor nodes include a low-power 802.15.4 radio operating in the 2.4GHz band, and use the Atmel Lightweight Mesh protocol to communicate. Data are recorded at a rate of 0.033 Hz (a sample every 30 seconds).

7.2.2.1 The Dataset

Data from 61 nodes were uploaded to the system. For each node the following streams were uploaded: humidity, temperature, illumination, pressure, battery, and alternative temperature. The stream lengths ranged from 1 week (19,958 samples) to 252 days (718,502 samples). The table below summarizes the dataset that was uploaded to the system.

Name	Sample Rate (Hz)	Number of Streams	Number of Days	Total number of samples	Total Size	Total Disk Size (+ DB overhead)
Humidity	0.033	61	8,439.78	24,282,258	185.25 MB	370.5 MB
Temperature	0.033	61	8,439.78	24,282,258	185.25 MB	370.5 MB
Illumination	0.033	61	8,439.78	24,282,258	185.25 MB	370.5 MB
Pressure	0.033	61	8,439.78	24,282,258	185.25 MB	370.5 MB
Battery	0.033	61	8,439.78	24,282,258	185.25 MB	370.5 MB
Alternative Temperature	0.033	61	8,439.78	24,282,258	185.25 MB	370.5 MB
Totals	NA	366	50,638.68	1,145,693,548	1.11 GB	2.22 GB

Table 4. The Tidmarsh study dataset summary

7.2.2.2 Prior methodology used to analyze the dataset

The deployed sensor nodes continuously transmit data to a wireless hub that is connected to the Internet. The data are stored in a PostgreSQL database under the following hierarchy (figure 8):

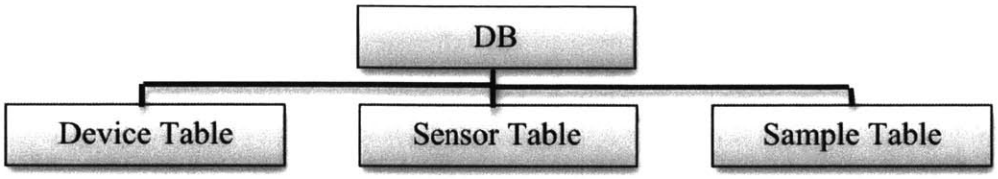


Figure 71. Tidmarsh database storage schema

The researchers used mostly Matlab and Python to analyze the data coming from the sensors.

7.2.3 Survey I results

Prior to working with the system, we asked the researchers several questions about their work processes: what tools they tend to use, their use of parallel and distributed frameworks, and their ideal analytics system. The written part of the survey questions can be found in Appendix A. A verbal interview followed the written survey in order to try and articulate some of the questions and gain a deeper understanding of the researchers approaches.

All of the participants used a combination of Python and Matlab for analyzing the data. Each of the researchers has their own programming environment and preferred code editors. Their scripts are mostly shared via email or some other file sharing mechanism. In many cases the scripts will initially not run on a different researcher`s environment as they may have external dependencies such as other packages that were installed by the user, or a specific version of the interpreter they use. There is an emphasis on writing scripts that are designed to solve the problem at hand and less on generic solutions that are extendable, and reusable. This is due both to the short-lived tenures of some of the researchers as well as the limited requirement of their analysis within a scope of a project,

Each of the research projects had a different approach to the collection and utilization of the sensor data. The SNAPSHOT researchers were predominantly concerned with collecting the data in its raw format, and then analyzing it upon the conclusion of the study cohort. The SNAPSHOT data were stored as files on a central file server. The Tidmarsh researcher's goal was to also provide a method to view the data as it was being collected, and therefore was concerned with the aggregation and filtering of the data during real-time as well as its analysis after a significant amount was collected. The Tidmarsh data were stored in a relational database.

In both studies the researchers were concerned with the quality of the collected data. Specifically in the case of wearable sensors that are not streaming the data in real-time, the researcher can only assess the quality at the conclusion of the study, or ask the participant to come in to the lab during the study for periodic data collection. Assessing the quality of the data is usually done by visualizing the raw signal and determining its quality. When asked whether this was something that could be automated, the response was that there are numerous corner

cases and that manually opening a random subset of a users files and viewing it can be more efficient than running scripts on the entire data set. Currently, there is active research on the automation of this process (Taylor et al., 2015).

In large datasets, the researchers acknowledged that significant amounts of time were spent on computation of features and on testing hypothesis. In some case, the computation could take several weeks. If the researchers found a problem with the scripts, additional features were required, or the hypothesis was changed, the entire process needed to be repeated. One of the questions that came to mind was whether the researchers were using parallel techniques in order to optimize the performance of their scripts. And if they were not, what was the reason? All of the researchers answered that they were not utilizing any parallel processing due to the following reasons:

-Parallel processing frameworks would require them to significantly alter their scripts and would take a significant amount of time and effort to debug.

-They would need to learn how to use parallel processing frameworks and were not willing to invest in a prolonged training, as they were required to present results.

-Utilizing multiple processors or threads on their own laptop was not an attractive option as the speed-up would only be marginal in their opinion.

-They did not have access to computation resources. A solution that could speed up their code significantly would require them to secure servers or VMs which would need to be budgeted.

-They lack the expertise to install and manage a cluster of computers themselves, and did not have a dedicated resource that would support them.

Clearly, parallel processing was not a tool that was important in their regular workflow.

Finally, we were interested in finding out what qualities would a "dream" analytics system possess. Below are their answers:

"Both raw and relevant features are sent from the device to a database. This can be accessed as instantaneously as possible so that questions can be formulated, asked, and

answered. The system would also highlight features that separated participants in ways that I am interested in. For example, if I am interested in health, the system would notify me if one of the features (or a combination) separated the participants into healthy and sick groups."

"Being able to fluidly move through large datasets and plot across different timescales, switching aggregation methods on the fly would be really useful. If I'm recording a measurement every several seconds and then want to look at a month's worth of data, obviously I don't want to plot every single point. Depending on what I'm looking for, the choice of aggregation method might be different. If I'm looking at overall temperatures across a site, aggregation by averaging is probably what I want. But if I'm trying to identify faulty sensors, I want to see the outliers that would otherwise be hidden by averaging. In many systems, aggregation is expensive and the parameters have to be decided up front. "

"I also often work with datasets that have many different sensors. It is useful to plot sensors for an entire site on the same axes to be able to identify which ones are producing values that are different or interesting. The number of sensors quickly exhausts the set of colors or line styles that can be easily differentiated, so being able to select a line on plotted data and figure out what sensor produced it would be very useful to me. One of the big challenges with scientific data in general is sharing between different research groups. Every researcher seems to use different formats and in particular methods of annotating what a given dataset actually contains. I know there are researchers who are interested in my data, but putting it into formats that can be easily shared becomes very impractical. (A CSV per sensor per reasonable length of time quickly turns into thousands of files, and how are the files named, how is the metadata/sensor location/etc attached to the files?) A system that would let other researchers browse through each others data and export what they want at the resolution they want (or just perform the analysis they want in the tool) would be fantastic."

"A software that locates the data on my computer and uploads it easily. Then asks me what kind of analysis I want to run. Based on my choice, it should show me a visualization user interface that can intelligently determine the relevant features and plot them for me side by side. Also calculate all the preliminary analysis like the ones I described above. And it should be

multi-modal too i.e. be able to handle different kinds of data points seamlessly. And then do the analysis very quickly."

From their answers it was obvious that the researchers were concerned with the speed at which they are able to obtain results. A second concern was data visualization – being able to plot the data at various timescales and aggregations. The researchers would also like the system to be capable of making intelligent suggestions such as determining a relevant feature that differentiates between cohorts.

7.2.4 Initial Experimental Results

During the experiment, we asked the users in both studies to provide us with examples of analytics that they executed on Tributary. In some cases, they also provided us with how long the same operations took using their previous traditional environments.

Study 1 – SNAPSHOT

1. **Hypothesis Testing: Do sleep storms at wake correlate with waking up feeling tired?**

   Cluster characteristics: 7 nodes, 64GB RAM per node, total of 45 cores
   Streams: The researcher loaded 145 EDA streams and 145 sleep streams. The EDA streams were recorded from a wearable sensor and the sleep streams were based on user self report of the time they fell asleep and awoke.
   Traditional execution time: 24 hours
   Tributary Load time: 2 hours for 2.7 Billion samples
   Tributary Execution time: 1.3 hours

   Total improvement: 727%

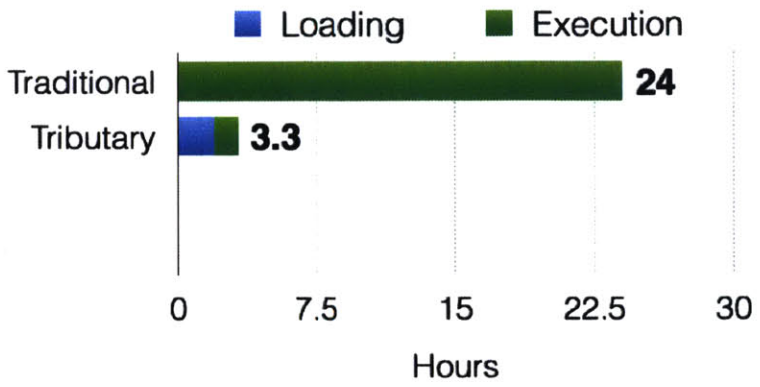


Figure 72. Hypothesis testing performance comparison

2. **Feature extraction: Extract EDA features needed for machine learning classification algorithm (72 features)**

Cluster characteristics: 7 nodes, 64GB RAM per node, total of 45 cores

Streams: The researcher loaded 145 EDA streams and 145 sleep streams. The EDA streams were recorded from a wearable sensor and the sleep streams were based on user self report of the time they fell asleep and awoke.

Traditional execution time: 72 hours

Tributary Load time: 2 hours for 2.7 Billion samples

Tributary Execution time: 1.5 hours

Total improvement: 2057%

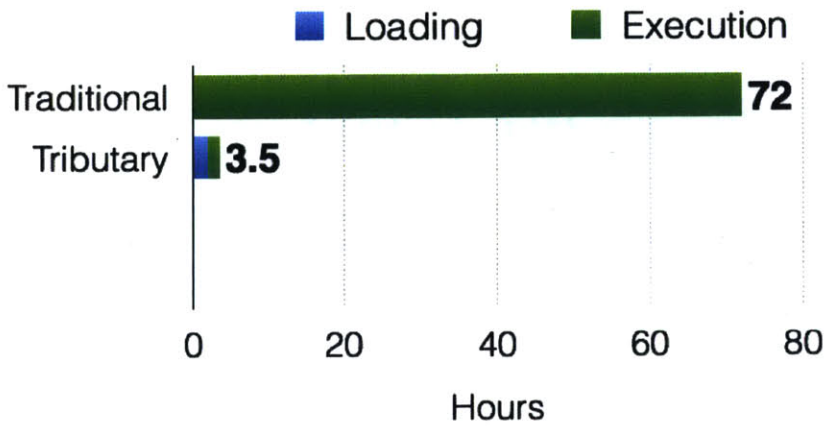


Figure 73. EDA Feature extraction performance comparison

3. **Feature Extraction: Accelerometer activity magnitude per second (minute window)**

Cluster characteristics: 11 nodes, 16GB RAM per node, total of 88 cores

Streams: The researcher loaded 47 Accelerometer streams. The Accelerometer streams were recorded from a wearable sensor.

Traditional execution time: 7.2 days

Tributary Load time: 36 minutes hours for 1 Billion samples

Tributary Execution time: 2 hours

Total improvement: 6646%

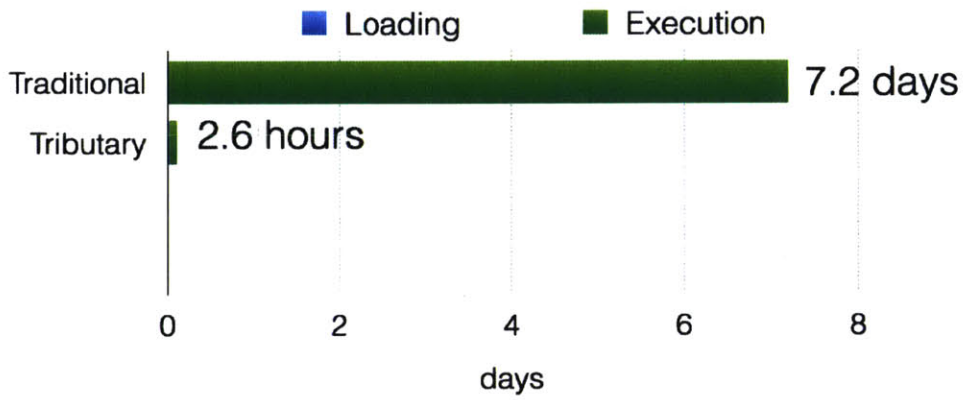


Figure 74. Feature extraction accelerometer performance comparison

Study 2- Tidmarsh

The researcher in the Tidmarsh study did not run an analysis with other tools prior to using Tributary, so we could not compare performance data for the scenarios. They provided us with a spatial mapping of the results overlaid onto an aerial photograph of the sensor node deployment.

1. Packet loss calculation (expected / received)

   Cluster characteristics: 5 nodes, 16GB RAM per node, total of 40 cores

   Streams: The researcher loaded 53 illumination streams.

   Traditional execution time: NA

   Tributary Load time: 15 minutes hours for 5 million samples (53 streams for 30 days)

   Tributary Execution time: 7.6 minutes

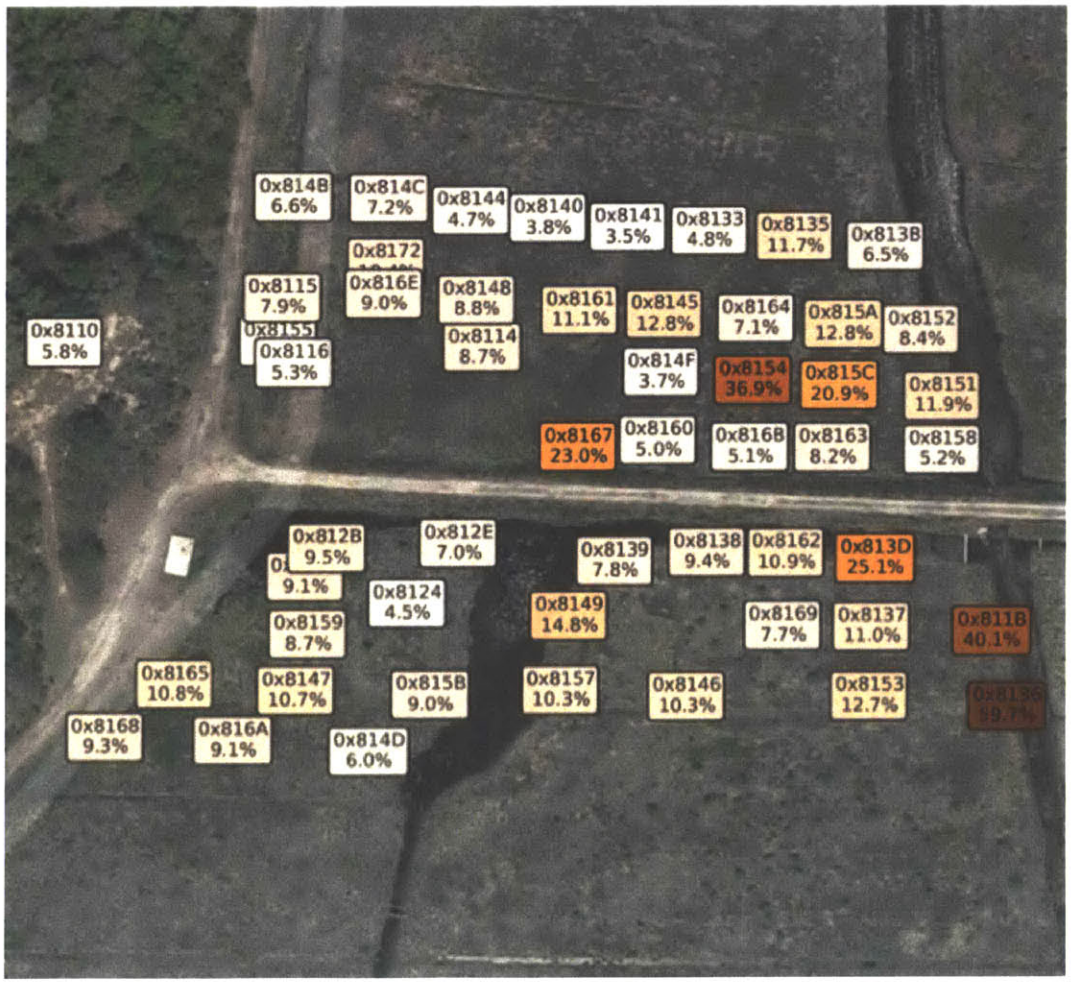


Figure 75. Packet loss rates overlaid on aerial view of sensor nodes. The two nodes on the bottom right that have the highest packet loss are located on a hill and therefor suffer from wireless connectivity problems

2. Average daily humidity (July)

Cluster characteristics: 5 nodes, 16GB RAM per node, total of 40 cores

Streams: The researcher loaded 53 humidity streams.

Traditional execution time: NA

Tributary Load time: 15 minutes hours for 5 million samples (53 streams for 30 days)

Tributary Execution time: 8 minutes

Figure 76. Average daily humidity overlaid on aerial view of sensor nodes

3. Hourly correlation between illumination and temperature hourly mean

   Cluster characteristics: 5 nodes, 16GB RAM per node, total of 40 cores

   Streams: The researcher loaded 53 illumination and 53 temperature streams.

   Traditional execution time: NA

   Tributary Load time: 30 minutes hours for 10 million samples (106 streams for 30 days)

   Tributary Execution time: 49.3 minutes

Figure 77. Hourly correlation between illumination and temperature hourly mean overlaid on aerial photo of sensor nodes. The sensor node in red produced a negative correlation because it was covered from external light. The internal light sensor was only exposed to an LED within the node that flashes when the node battery is discharging. Discharging occurs mainly at night, because the nodes are solar powered.

7.2.5 Survey II and SUS results

At the end of the study we performed an ethnographic interview to obtain qualitative information on the usability of the new tool. In the first part of the interview we asked the participants to fill in two sets of surveys. The first set of questions was asked to assess system usability. We used the system usability scale (SUS) (Brooke, 1996), which is a simple, ten-item scale giving a global view of subjective assessments of usability. SUS Questions are answered on a 7-point Likert scale and yields a single number between 0-100 representing a composite measure of the overall usability of the system.

The second set of questions was used to assess the user's experience during the use of the system. In the second part of the interview we asked the participants both open-ended questions and rating questions regarding their personal experience during the study. We wanted to determine which features the users found useful, which features were difficult to user or provide little value, and what features they would have liked to see implemented.

The system usability score ranged between 72.5 and 77.5 and the average score was 74.16 (standard deviation = 2.88). The two items that scored lowest in the SUS were the following items:

1. I would imagine that most people would learn to use this system very quickly
2. I felt very confident using the system

When asked about the first item, the researchers noted that although they themselves did not have much trouble learning to use the new system, other users less versed with data analysis may have require a steeper learning curve. When asked about the second item, the researchers all felt that because the system was new and unproven it would require additional time for them to trust the system's results blindly.

I compiled the users answers to the rating questions of the second survey in the table below (table 5):

Question	**Average**	**Standard Deviation**
Estimate the percent of the analysis you currently do that can be replaced by this tool? (0-100%)	66.1%	5.77
Was it easier or harder to explore the data using the new tool? (1- extremely easy, 10 – extremely hard)	3.66	0.57
Was it easier or harder to manipulate the data using the new tool? (1- extremely easy, 10 – extremely hard)	3.66	1.52
Was it easier or harder to asses the quality of the data using the new tool? (0- extremely easy, 10 – extremely hard)	2.66	1.15
Was is it easier or harder to gain new insights from the data using the new tool? (0- extremely easy, 10 – extremely hard)	4.66	0.57

Table 5. Survey 2 - Quantitative question responses

The above table suggests that the users found the system useful in exploring and manipulating data and better than their existing tools in most cases.

We asked the researchers what they liked most about the new tool:

-*"I didn't have to re-write code to load data when I wanted to change the type of data (e.g., EDA vs. temperature). I also really loved that it was a lot faster than my computer could do. I could also see all of the data and very quickly search it. Something not even imaginable on my own computer."*

-*"The analysis was surprisingly FAST!!! "*

-*"I don't otherwise have a good workflow for doing this kind of processing on my entire data set, on a cluster or otherwise. The tool readily provided me with both the capability to query the data of interest and perform operations on it. While I could have written scripts to fetch data from my database and perform analysis, I'd probably be writing scripts to address particular hypotheses that wouldn't necessarily be very reusable, and would have to make the decision as to whether the extra time to write code to run on a distributed platform would*

outweigh the extra processing time of running it single-threaded. The tool provided me with those capabilities without a lot of work."

We also asked them if there was a feature that was particularly delightful. Below are the responses:

-*"The speed. So fast. Also the search was surprisingly delightful. I was able to explore and see(!) entire data sets very quickly and come up with hypothesis that I never would have noticed if this weren't possible."*

-*"I really like the loading tool. It helped me visualize the data a lot better."*

-*"I'd say one of my favorite things was being able to start analysis that was going to take some time, log out, go home, and check on the result just by logging into a web browser wherever I happened to be."*

When asked what the user least liked about the new tool one researcher said that it was hard to debug the user operators, as the error messages were difficult to comprehend, and there were problems with error line numbers. Another said that when just starting to work with the system it was difficult to anticipate how long an operation would take. This became less of an issue after knowing the tool better.

We asked if there were any part that seemed stupid / clever? For example requiring more steps than the user needed or enabling them to do something that they hadn't thought of before? One researcher said that the liked how the visualization enabled them to see the logical flow of operators: "It was really helpful in think about how my analysis was set up and where I could make modifications". Another research said that the system did require some knowledge of how the data would be distributed in the cluster and that made it a bit more difficult than just programing for a single-threaded application, but that was significantly better than dealing with building a parallel processing programming using one of the standard frameworks.

We asked what the users what they thought could be added to the tool. These were some of the features they thought would improve the tool:

-An option to test the operator on a very small piece of the dataset would help debug the operators

-Provide access to various stream meta-data fields within the operator, to automatically populate some variables. For example, providing the starting timestamp of each block would enable to perform sanity tests on sleep classification.

-Having built in machine learning. For example, an operator that uses an SVM to classify output or a simple regression tool.

- Provide a button to export the visualization plots to files on the users computer

- Ability to compare between computation nodes

-Showing in each session what query loaded the data. When utilizing multiple session it is easy to forget what is load in each session

The researchers were asked whether they would utilize the sharing of data in the system with other researchers. The acknowledge that this was an important use case:

-"Yes, if they were collaborators. It would be much easier to share data via a system like this instead of Dropbox because the data is so massive."

-"I would like to share my data with my collaborators using the system. Not sure about every single user of the system"

-"Absolutely. Sharing data sets with other researchers in the field, especially for large data sets, is always somewhat of a challenge. Having an easy-to-use graphical interface that allows other researchers to query and process my data would be great. The alternatives are forcing data into common formats (CSV, Matlab files) and trying to figure out how to communicate all of the relevant metadata; there are many proposed formats for data interchange but none work very well. Sharing within the analysis tool itself alleviates some of those issues."

The responses regarding sharing if operators with other researchers were mixed:

-"YES! Then they could check if the operators made sense or if I had a bug in any of them. I would love it if people who reviewed my analysis or papers could do similar analysis on their own data to see if it was repeatable."

-"It would depend on the operator and what it's used for. I can probably share basic operators."

-"Maybe, if I had complex operators that I thought would be generally useful. For many simpler things it might not be worth the time to document the operator well enough that others could use it. Sometimes I might write an operator that has constants in it that only make sense for running on a one-off query, where it wouldn't necessarily even be reusable by myself."

It seems that in order to make sharing viable, the users feel that they would need to document the operators and that may entail effort.

The users were asked if they utilized the data exporting functionality. The users found this functionality particularly useful for two reasons. The first was the ability to use their own visualization tools to produce various plots of the data. The second was the ability to compare results to some other data that was not imported to the system.

In summary, it appears that the system achieved most of its intended design goals: it enabled users to interact with large-scale time series datasets much faster than they could previously. Once they understood the paradigm, they could both query and manipulate the data utilizing parallel computation without the added complexities of modern distributed computation frameworks. This "learning" period took no more than several hours.

8 Conclusions

8.1 Thesis Contributions

The goal of this thesis was to explore and extend the affordances of distributed computing for interactivity and pliability of large-scale time series data sets.

In this thesis I have made the following contributions:

1. I have presented a detailed analysis and study of the biophysiological signal processing pipeline
2. I have introduced a theoretical framework and nomenclature which serves as an abstraction for distributed processing of time series data
3. I have presented An architecture for distributed storage and processing of bio-physiological signals
4. I have implemented the framework and architecture and built a multi-user system, Tributary, which provides
    a. A distributed processing engine
    b. A distributed Storage platform
    c. An interface for interacting with very large sensor data sets
5. I have validated the system and presented a detailed chronicle of the design considerations, implementation, testing and its evaluation.
6. I have designed a novel interface for exploring an interacting with large-scale sensor data sets that facilitates human-in-the loop computation. The interface provides the ability for a user to compare between multiple iterations over the same input and select past iterations for computation as well.

8.2 Future Work

Below, I have listed several important features that may increase the utility of the system:

8.2.1 Stream data load frequency analysis and optimization

The load time of a dataset from disks to RAM is often a significant portion of the entire execution time. From analyzing the system use patterns, it became clear that researchers load the same group of streams while working on a specific study. Therefore, a possible optimization would be to maintain a record of the most "utilized" streams, and maintain those in RAM. A Least Recently Used (LRU) cache eviction mechanism can be implemented: streams that are less used are evicted from RAM. In case RAM is limited, it will be possible to maintain a serialized version of the streams on disk. It is faster to read the serialized streams from disk as opposed to retrieving them using a database query.

8.2.2 Streaming: running operators on real-time data

The current implementation of Tributary supports running of operators only on streams that have been imported to the system and stored in the database. It is possible to create a new type of operators that will execute on data as it's been received by the system. These operators are called streaming operators. An example use case could be running a filter on the incoming samples in order to remove artifacts or calculate various aggregations for powering a real time dashboard.

8.2.3 Visualization customization operators

One of the feedbacks that we got from the user study was that the users would like to have more control over the visualizations. Currently, a user can define the output of an operator to be one of the following: curve, bar plot, scatter plot, or distribution. If users would like a different type of visualization, they can export the data over HTTP into any other visualization

environment such as Python or Matlab. A more efficient way would be to add a visualization operator that can be implanted by the user. For example, the user could implement a pie chart operator that would be applied to the output of a previous operator. The user could utilize a visualization package of their choice, such as Python Matplotlib ("matplotlib: python plotting — Matplotlib 1.5.0 documentation," n.d.), which would allow flexibility.

8.2.4 Storage and retrieval of operator pipelines

In case where users would like to repeat a set of operators, it would be possible to create an operator pipeline by saving a group of previously applied operators in the order of their application as a new operator. This pipeline can be applied as a single operator, without requiring the user to apply multiple operations, one after the other.

8.2.5 External Data source integration (EMR, PHRs, etc.)

Currently, the user can import data into the system by uploading files containing the sensor sample data. If the user would like to utilize a stream that already exists in an external system, such as an Electronic Medical Record (EMR), then the users are required to export the data from such other system and then import it into Tributary. Instead, it would be convenient to integrate between Tributary and these external systems so that streams could be directly pulled from them. This would enable utilizing clinical information with greater ease and find possible correlations with health outcomes.

8.3 Outlook

This thesis has demonstrated the utility of creating an abstraction for distributed computation of time series data sets, as well as providing a platform for storage and analytics of large-scale time series data sets. In order to harness the power of advanced warehouse scale computing platforms for research, we must bridge the gap between traditional workflows and massively parallel processing by abstracting out the mechanics of controlling and utilizing storage, memory, networks and processors. With the advent of sensors and connected devices, I envision this platform to be applicable to a wide variety of fields: automotive, energy, health, and manufacturing to name a few.

Bibliography

Ahlberg, C., & Shneiderman, B. (1994). Visual information seeking. In *Proceedings of the SIGCHI conference on Human factors in computing systems celebrating interdependence - CHI '94* (pp. 313–317). New York, New York, USA: ACM Press. doi:10.1145/191666.191775

Apache™ Hadoop - reliable, scalable, distributed computing. (n.d.). Retrieved from http://hadoop.apache.org/

AWS | Amazon Elastic Compute Cloud (EC2) - Scalable Cloud Hosting. (n.d.). Retrieved May 26, 2014, from http://aws.amazon.com/ec2/

Azure: Microsoft's Cloud Platform | Cloud Hosting | Cloud Services. (n.d.). Retrieved May 26, 2014, from http://azure.microsoft.com/en-us/

Bagby, R. M., Parker, J. D. A., & Taylor, G. J. (1994). The twenty-item Toronto Alexithymia scale—I. Item selection and cross-validation of the factor structure. *Journal of Psychosomatic Research, 38*(1), 23–32. doi:10.1016/0022-3999(94)90005-1

Barroso, L. A., & Hölzle, U. (2009). *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. Synthesis Lectures on Computer Architecture* (Vol. 4). Morgan & Claypool. doi:10.2200/S00193ED1V01Y200905CAC006

Berard, A., & Hebrail, G. (2013). Searching time series with Hadoop in an electric power company. In *Proceedings of the 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining Algorithms, Systems, Programming Models and Applications - BigMine '13* (pp. 15–22). doi:10.1145/2501221.2501224

Bresnick, J. (2014). What are the barriers to clinical analytics, big data success? | HealthITAnalytics.com. *Health IT Analytics*. Retrieved October 20, 2014, from http://healthitanalytics.com/2014/07/30/what-are-the-barriers-to-clinical-analytics-big-data-success/

Brooke, J. (1996). SUS-A quick and dirty usability scale. In P. Jordan, B. Thomas, B. Weerdmeester, & A. McClelland (Eds.), *Usability evaluation in industry* (pp. 189–194). London: Taylor and Francis.

Chatfield, C. (1984). *The Analysis of Time Series: An Introduction.* Boston, MA: Springer US. doi:10.1007/978-1-4899-2921-1

Ciccarese, P., & Larizza, C. (2006). A framework for temporal data processing and abstractions. *AMIA ... Annual Symposium Proceedings / AMIA Symposium. AMIA Symposium*, 146–50.

Retrieved from http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=1839476&tool=pmcentrez&rendertype=abstract

Cloud Computing & Cloud Hosting Services — Google Cloud Platform. (n.d.). Retrieved May 26, 2014, from https://cloud.google.com/

Collier, R. (2009). Rapidly rising clinical trial costs worry researchers. *CMAJ : Canadian Medical Association Journal = Journal de l'Association Medicale Canadienne*, *180*(3), 277–8. doi:10.1503/cmaj.082041

Covinsky, K. E. (2013). The incremental nature of clinical research: comment on "The association of aspirin use with age-related macular degeneration". *JAMA Internal Medicine*, *173*(4), 266. doi:10.1001/jamainternmed.2013.2790

DateTimeFormat (Joda time 2.2 API). (n.d.). Retrieved October 31, 2015, from http://joda-time.sourceforge.net/apidocs/org/joda/time/format/DateTimeFormat.html

Dawber, T. R., Meadors, G. F., & Moore, F. E. (1951). Epidemiological approaches to heart disease: the Framingham Study. *American Journal of Public Health and the Nation's Health*, *41*(3), 279–81. Retrieved from http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=1525365&tool=pmcentrez&rendertype=abstract

Dean, J., & Ghemawat, S. (2004). MapReduce: simplified data processing on large clusters. In *OSDI 2004* (pp. 1–13). Retrieved from http://dl.acm.org/citation.cfm?id=1327492

dos Santos, L. D. P., da Silva, A., Jacquin, B., Picard, M.-L., Worms, D., & Bernard, C. (2012). Massive Smart Meter Data Storage and Processing on top of Hadoop. In *Vldb 2012*. Istanbul, Turkey.

Ewen, S., Schelter, S., Tzoumas, K., Warneke, D., & Markl, V. (2013). Iterative parallel data processing with stratosphere. In *Proceedings of the 2013 international conference on Management of data - SIGMOD '13* (p. 1053). New York, New York, USA: ACM Press. doi:10.1145/2463676.2463693

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. University of California, Irvine. Retrieved from http://dl.acm.org/citation.cfm?id=932295

Fitchard, K. (2013). Ericsson: Global smartphone penetration will reach 60% in 2019 — Tech News and Analysis. Retrieved October 20, 2014, from https://gigaom.com/2013/11/11/ericsson-global-smartphone-penetration-will-reach-60-in-2019/

Garrett, J. J. (2010). *Elements of User Experience,The: User-Centered Design for the Web and Beyond.* Pearson Education. Retrieved from https://books.google.com/books?hl=en&lr=&id=9QC6r5OzCpUC&pgis=1

GridGain = In-Memory Computing. (n.d.). Retrieved from http://www.gridgain.com/

Hall, M., Kirby, R. M., Li, F., Meyer, M., Pascucci, V., Phillips, J. M., ... Venkatasubramanian, S. (2013). Rethinking Abstractions for Big Data: Why, Where, How, and What, (1), 1–8. Retrieved from http://arxiv.org/abs/1306.3295

Hartl, M. (2012). *Ruby on Rails Tutorial: Learn Web Development with Rails.* Addison-Wesley Professional. Retrieved from http://dl.acm.org/citation.cfm?id=2392648

Hazelcast | In-Memory Data Grid. (n.d.). Retrieved from http://www.hazelcast.com/

Iverson, K. E. (1962). *A programming language.* Wiley. Retrieved from https://books.google.com/books/about/A_programming_language.html?id=zR81AAAAIAAJ&pgis=1

KairosDB - Fast scalable time series database. (n.d.). Retrieved October 17, 2014, from https://github.com/kairosdb/kairosdb

Karwowski, W. (2006). *International Encyclopedia of Ergonomics and Human Factors, Second Edition - 3 Volume Set.* (W. Karwowski, Ed.). CRC Press. doi:10.1201/9780849375477

Kushner, D. (2011). The Making of Arduino - IEEE Spectrum. *IEEE Spectrum.* Retrieved October 26, 2015, from http://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino

Lakshman, A., & Malik, P. (2010). Cassandra - A Decentralized Structured Storage System. *ACM SIGOPS Operating Systems Review*, *44*(2), 35–39. doi:10.1145/1773912.1773922

Lederman, S. (2014). The Evolving Role Of Drug Mechanism Of Action In Drug Discovery And Development. *Life Science Leader.* Retrieved from http://www.lifescienceleader.com/doc/the-evolving-role-of-drug-mechanism-of-action-in-drug-discovery-and-development-0001

Leland, K. (2013). Study Uses Mobile Technology to Help Predict and Prevent Heart Disease | UC San Francisco. Retrieved October 26, 2015, from http://www.ucsf.edu/news/2013/03/13695/study-uses-mobile-technology-help-predict-and-prevent-heart-disease

Lohr, S. (2014). For Big-Data Scientists, "Janitor Work" Is Key Hurdle to Insights - NYTimes.com. *NYTimes.com.* Retrieved October 17, 2014, from http://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html?_r=0

Lowgren, J., & Stolterman, E. (2007). *Thoughtful Interaction Design: A Design Perspective on Information Technology*. The MIT Press.

Lyons, R. G. (2010). *Understanding Digital Signal Processing* (Vol. 1). Pearson Education. Retrieved from https://books.google.com/books?id=UBU7Y2tpwWUC&pgis=1

Maeda, J. (2001). *Design by Numbers*. MIT Press.

matplotlib: python plotting — Matplotlib 1.5.0 documentation. (n.d.). Retrieved October 31, 2015, from http://matplotlib.org/

McKenna, T. M., Bawa, G., Kumar, K., & Reifman, J. (2007). The physiology analysis system: an integrated approach for warehousing, management and analysis of time-series physiology data. *Computer Methods and Programs in Biomedicine*, *86*(1), 62–72. doi:10.1016/j.cmpb.2007.01.003

Middleton, P., Kjeldsen, P., & Tully, J. (2013). *Forecast: The Internet of Things, Worldwide, 2013.*

Nielsen, L. (2012). *Personas - User Focused Design*. Springer Science & Business Media.

Open Source Hardware Summit Speech 2011. (2011). Retrieved October 28, 2015, from http://www.slideshare.net/arduinoteam/open-source-hardware-summit-speech-2011

OpenTSDB - A Distributed, Scalable Monitoring System. (n.d.). Retrieved from http://opentsdb.net/

Orsini, L. (2014). Arduino's Massimo Banzi: How We Helped Make The Maker Movement - ReadWrite. Retrieved October 28, 2015, from http://readwrite.com/2014/05/12/arduino-massimo-banzi-diy-electronics-hardware-hacking-builders

Reas, C. (2015). Casey Reas at p5js Diversity. *Opentranscripts*. Retrieved October 28, 2015, from http://opentranscripts.org/transcript/casey-reas-p5jscon/

Reas, C., & Fry, B. (2007). *Processing: A Programming Handbook for Visual Designers and Artists*. MIT Press.

Rogers, E. M. (1995). *Diffusion of Innovations, Fourth Edition*. Free Press. Retrieved from http://www.amazon.com/Diffusion-Innovations-Fourth-Edition-Everett/dp/0029266718

Sazonov, E., & Neuman, M. (2014). *Wearable Sensors - Fundamentals, Implementation and Applications* (1st ed.). Academic Press.

Scott, M. L. (2009). *Programming Language Pragmatics. Programming Language Pragmatics*. Elsevier. doi:10.1016/B978-0-12-374514-9.00040-9

Shneiderman, B. (1996). The eyes have it: a task by data type taxonomy for information

visualizations. *Proceedings 1996 IEEE Symposium on Visual Languages*, 336–343. doi:10.1109/VL.1996.545307

Stonebraker, M., Becla, J., Dewitt, D., Lim, K.-T., Maier, D., Ratzesberger, O., & Zdonik, S. (2009). Requirements for Science Data Bases and SciDB. *4th Biennial Conference on Innovative Data Systems Research CIDR'09*, 173184. doi:10.1.1.145.1567

Storm, distributed and fault-tolerant realtime computation. (n.d.). Retrieved October 17, 2014, from https://storm.incubator.apache.org/

Sunderam, V. S. (1990). PVM: A framework for parallel distributed computing. *Concurrency: Practice and Experience*, *2*(4), 315–339. doi:10.1002/cpe.4330020404

Taylor, S., Jaques, N., Chen, W., Fedor, S., Sano, A., & Picard, R. (2015). Automatic Identification of Artifacts in Electrodermal Activity Data. In *International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. Milan. Retrieved from http://affect.media.mit.edu/pdfs/15.Taylor-Jaques-et-al-ArtifactDetectionEDA.pdf

TempoIQ. (n.d.). Retrieved October 17, 2014, from https://www.tempoiq.com/?r=1

Terracotta | In-Memory Data Management for the Enterprise. (n.d.). Retrieved from http://terracotta.org/

Whitney, A., & Shasha, D. (2001). Lots o'Ticks. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data - SIGMOD '01* (Vol. 30, p. 617). New York, New York, USA: ACM Press. doi:10.1145/375663.375783

Yamamoto, Y., & Nakano, J. (2001). *Distributed computing in a time series analysis system*. Humboldt University of Berlin, Interdisciplinary Research Project 373: Quantification and Simulation of Economic Processes. Retrieved from http://econpapers.repec.org/RePEc:zbw:sfb373:200176

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: cluster computing with working sets. In *HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing* (p. 10). USENIX Association.

Zhang, Y., Kersten, M., Ivanova, M., & Nes, N. (2011). SciQL, Bridging the Gap between Science and Relational DBMS. In *Proceedings of the 15th Symposium on International Database Engineering & Applications - IDEAS '11* (p. 124). New York, New York, USA: ACM Press. doi:10.1145/2076623.2076639

APPENDICES

APPENDIX A – Survey 1

Tributary - Massive Time-series Analytics - Survey 1

* Required

Participant ID number *

Describe the steps taken when processing data from a study. Steps may include - saving to files / database, converting file formats, filtering, normalization, etc. Try to be as detailed as possible. *

For each of the steps you described above, how long does each step typically take? You may specify the times for a certain amount of data collected if that is more convenient. *

Which of these steps takes the longest time? why ? *

If you had to describe your "dream" sensor / survey data analysis system, what would it be? Assume that there are no engineering, computation, usability limitations and that anything is possible. *

Additional comments

APPENDIX B – Survey 2

System Usability Survey (SUS)

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

Using SUS

The SU scale is generally used after the respondent has had an opportunity to use the system being evaluated, but before any debriefing or discussion takes place. Respondents should be asked to record their immediate response to each item, rather than thinking about items for a long time.

All items should be checked. If a respondent feels that they cannot respond to a particular item, they should mark the centre point of the scale.

Scoring SUS

SUS yields a single number representing a composite measure of the overall usability of the system being studied. Note that scores for individual items are not meaningful on their own.

To calculate the SUS score, first sum the score contributions from each item. Each item's score contribution will range from 0 to 4. For items 1,3,5,7,and 9 the score contribution is the scale position minus 1. For items 2,4,6,8 and 10, the contribution is 5 minus the scale position. Multiply the sum of the scores by 2.5 to obtain the overall value of SU.

SUS scores have a range of 0 to 100.

The following section gives an example of a scored SU scale.

APPENDIX C – Survey 3

Tributary - Massive Time Series Analytics: Survey 3

Questions regarding your interaction with the tributary system

* Required

Participant ID Number *

How often did the new tool crash or hang ? *

Other than the above, what did you most dislike about the new tool ? Why ? *

What did you most like about the new tool? Why ? *

Estimate the percent of the analysis you currently do that can be replaced by this tool *

0 1 2 3 4 5 6 7 8 9 10

None ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ All

If you answered the above question with less than 10, please explain what would prevent you from using this tool exclusively? *

In your opinion, how could the new tool be improved? What features would you add to it ? *

Was it easier or harder to explore the data using the new tool ? *

1 2 3 4 5 6 7 8 9 10

Extremely Easy ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ Extremely Hard

Was it easier or harder to manipulate the data using the new tool ? *

1 2 3 4 5 6 7 8 9 10

Extremely Easy ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ Extremely Hard

Was it easier or harder to asses the quality of the data using the new tool ? *

1 2 3 4 5 6 7 8 9 10

Extremely Easy ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ Extremely Hard

Was is it easier or harder to gain new insights from the data using the new tool ? *

1 2 3 4 5 6 7 8 9 10

Extremely Easy ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ Extremely Hard

If your answer to any of the above 4 questions was "Hard", please give an example *

If your answer to any of the above 4 questions was "Easy", please give an example *

Would you utilize the option to share data with other users of the system? Why or why not? *

Would you utilize the option to share operators you wrote with other users of the system ? Why or why not? *

Did you utilize exporting of the data out of the system (for continuing analysis using a different tool, backup, or visualization) ? Why or why not? *

Was any particular feature delightful ? *

What features saved you time? *

Were there any part that seemed stupid / clever? For example required more steps than you needed or enabled you to do something you hadn't thought of before? *

Additional comments about your interaction with the system ?

APPENDIX D – REST endpoints

GET /test	A test endpoint
GET /search	Searches across all streams in the database. Receives a query string as a parameter, returns
GET /operators	Retrieve all operators of a specific type. Receives one of the following operator types: transformation, aggregation, combination, or selection as a parameter and returns all of the operators of that type. Each operator is a JSON structure with the following fields: id, name, type, description, owner, args, creation_date, modification_date, source_code, language, input_type, output_type, plots, and plot_type.
GET /operators/:id	Receives an operator ID as a parameter and returns all operator fields as a JSON structure. These include id, name, type, description, owner, args, creation_date, modification_date, source_code, language, input_type, output_type, plots, and plot_type.
POST /operators	Create a new operator.
PUT /operators/:id	Update an existing operator.
DELETE /operators/:id	Delete an existing operator.
POST /sessions	Adds the result of a query to the list of streams to be loaded into a session.
GET /sessions	Get all existing sessions for a user.

POST /nodes	Apply an operation to a node in a session.
GET /nodes/:id	Called when a session is created is first created. Loads the selected streams of the session into the RAM of the cluster. Returns the stream previews and the node metadata.
PUT /nodes/:id	Selects a node from a group of sibling nodes to be on top.
GET /sensors	Returns a list of currently supported sensors.
GET /streams	Returns node metadata and all stream previews of the node.
POST /login	Authenticate a user. Receives "email" and "password" as parameters. Returns a token that can be stored by the browser in a cookie.
DELETE /nodes/:id	Deletes a node with the specified ID.
DELETE /sessions/:id	Deletes an entire session with the specified ID.
GET /sessions/:query/progress	Retrieves the current progress of the last operator that was applied within a session. Receives the session ID as a parameter and returns the progress as a percentage.
POST /upload	Upload a data file containing either a single stream or an archive of multiple streams. Receives the file as well as additional stream meta data parameters such as sample rate, timestamp format, study id, group id, source id, and sensor type.
GET /streamids	Return all of the stream-ids within a node. Receives the node-id as input and returns a list of stream-ids.

GET /stream	Returns all of the streams data. This is used when the user whishes to view all of the streams within an entire node. Receives the node-id as input and returns all of the streams and their data. The streams are subsampled to enable viewing the data in a web browser
GET /node	Returns entire raw data (un-sampled) for a specific node. Often used after an aggregation operation has reduced the size of the data significantly.

APPENDIX E – Support Timestamp Formats

The following are taken from the Joda time Java package documentation.

Symbol	Meaning	Presentation	Examples
G	era	text	AD
C	century of era (>=0)	number	20
Y	year of era (>=0)	year	1996
x	weekyear	year	1996
w	week of weekyear	number	27
e	day of week	number	2
E	day of week	text	Tuesday; Tue
y	year	year	1996
D	day of year	number	189
M	month of year	month	July; Jul; 07
d	day of month	number	10
a	halfday of day	text	PM
K	hour of halfday (0~11)	number	0
h	clockhour of halfday (1~12)	number	12
H	hour of day (0~23)	number	0
k	clockhour of day (1~24)	number	24
m	minute of hour	number	30
s	second of minute	number	55
S	fraction of second	number	978
z	time zone	text	Pacific Standard Time; PST
Z	time zone offset/id	zone	-0800; -08:00; America/Los_Angeles
'	escape for text	delimiter	
''	single quote	literal	'