

University of Nevada, Reno

Keystone: A Streaming Data Management Model for the Environmental Sciences

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in Computer Science and Engineering

by

Connor Francis Scully-Allison

Dr. Sergiu M. Dascalu, Thesis Co-Advisor
&
Dr. Frederick C. Harris, Jr., Thesis Co-Advisor

May, 2019

© by Connor Scully-Allison
All Rights Reserved



THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

Connor F. Scully-Allison

Entitled

Keystone: A Streaming Data Management Model for the
Environmental Sciences

be accepted in partial fulfillment of the
requirements for the degree of

Master of Science

Dr. Sergiu M. Dascalu , Advisor

Dr. Frederick C. Harris Jr. , Co-advisor

Dr. Scotty Strachan , Graduate School Representative

David W. Zeh, Ph.D., Dean, Graduate School

May-2019

Abstract

In this thesis we explore a problem facing contemporary data management in the earth and environmental sciences: effective production of uniform and quality data products which keeps pace with the volume and velocity of continuous data collection. The process of creating a quality data product is non-trivial and this thesis explores in detail what knowledge is required to automate this process for emerging mid-scale efforts and what prior attempts have been made towards this goal.

Furthermore, we propose a model which can be used to develop a mid-scale data product pipeline in the earth and environmental sciences: Keystone. Specifically, by automating Quality Assurance, Quality Control and Data Repair processes, data products can be created at a rate that keeps pace with the production of data itself.

To prove the effectiveness of this model, three software applications that fulfilled each of the key roles suggested by the Keystone model were conceived, implemented and validated individually. These three applications are the NRDC Quality Assurance Application, the Near Real Time Autonomous Quality Control Application (NRAQC), and the Improved Robust and Sparse Fuzzy K Means (iRSFKM) imputation algorithm. Respectively, they provide the functionalities of metadata management and binding through a multi-platform mobile application; automated data quality control with the help of a dynamic web application; and rapid data imputation for data repair. The latter leverages multi-gpu processing to add significant speed to a high accuracy algorithm.

The NRDC Quality Assurance application was validated with the aid of a directed user survey which was disseminated among environmental scientist members of the Earth Science Information Partners organization. An analysis of these surveys indicated that the NRDC Quality Assurance application addresses many significant gaps in the area of metadata binding and creation with many respondents still recording metadata on pen and paper and taking between hours to weeks to digitize their metadata.

The efficacy of the NRAQC application was demonstrated through a case study

where nearly a million data points were batch tested according to various user configured metrics. The NRAQC system consistently flagged the same data points on the same streams over the course of five iterations, with an average testing time of 134.24 seconds per testing iteration. Specifically, in each iteration, the NRAQC system identified 1946 repeat values, 365 missing values, and 131 out-of-range values.

Finally, we demonstrated the effectiveness of our iRSFKM algorithm and implementation with multiple experiments, clustering real environmental sensor data. These experiments showed that the our improved multi-GPU implementation is significantly faster than sequential implementations with 185 times speedup over eight GPUs. Experiments also indicated greater than 300 times increase in throughput with eight GPUs and 95% efficiency with two GPUs compared to one.

The overall model itself was validated through a discussion of how effectively these software solutions worked in tandem to produce a final data product in a primarily automated fashion.

Dedication

I dedicate this thesis to my mother, Mo Scully and my father, David Allison; two individuals whose inexhaustible and unconditional love and support has been invaluable in helping me accomplish this achievement.

Acknowledgments

I would sincerely like to thank my thesis committee, Dr. Dascalu, Dr. Harris, and Dr. Strachan, for reviewing this absurdly long document and taking the time to guide me through the ins and outs of professional academia. They have, each of them, been invaluable mentors and I would not be here today without their guidance and help.

I would like to additionally thank my colleagues who provided me significant support through the many up and downs of my tenure here at UNR. Vinh Le, Eric Fritzing, Hannah Munoz, Rui Wu and Chase Carthen. I sincerely thank you for all the time you took to make me a better student and academic. Additionally, I would like to thank all my friends old and new for their support throughout this long journey of mine from liberal artist to computer scientist. Thank you all for not saying I was crazy to attempt such a thing.

Finally I would like to thank my family. My mother and father for providing me with significant financial and emotional support throughout my Masters degree. My sisters, Geordie and Sophie, and my brother, Levi, who are perennially unafraid to call me the smartest of us siblings, and who are going to be far more successful than myself. And my girlfriend, Jewel Lambert, who has done nothing but enable me to be the best person I can possibly be.

This material is based on work supported in part by the National Science Foundation under grant numbers IIA-1329469 and IIA-1301726. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation. Additionally, this work is partially based on funding provided by the ESIP Community with support from the National Aeronautics and Space Administration (NASA), National Oceanic and Atmospheric Administration (NOAA) and the United States Geologic Survey (USGS). Finally, this work is partially based on funding provided through the Consortium of Universities for the Advancement of Hydrologic Science Hydroinformatics Innovation Fellowship.

Contents

Abstract	i
Dedication	iii
Acknowledgments	iv
List of Tables	viii
List of Figures	ix
Glossary	xiii
1 Introduction	1
1.1 The Humanity of data	1
1.2 The importance of data	2
1.3 The Purpose of This Document	4
2 Background & Related Work	6
2.1 Data Management	6
2.1.1 FAIR Data Principles	7
2.1.2 Data Management in the Earth Sciences	10
2.2 Earth Science Information Partners	12
2.3 The NRDC	13
2.3.1 History	13
2.3.2 Position as a Data Management Platform	13
2.4 Earth Science Data	14
2.4.1 Geospatial Data	15
2.4.2 Time Series Data	16
2.5 Quality Assurance and Quality Control	18
2.5.1 Quality Assurance	19
2.5.2 Quality Control	22
2.6 Data Imputation	25
2.6.1 Unsupervised Machine Learning	25
2.6.2 Clustering and Convex Optimization	27
2.6.3 Data Imputation Methods	29
2.7 GPU Programming	30
2.8 Critical Tools and Technologies	32
2.8.1 Containerization	32
2.8.2 OWL & RDF	34
2.8.3 Ionic	36
2.8.4 Python	36
2.8.5 Flask	37
2.8.6 Angular	38
2.8.7 CVXPY	39
2.8.8 CVXGEN	40
2.8.9 PyCUDA	40
3 Data Quality Model	41
3.1 The Keystone Model	41
3.2 Quality Assurance	44
3.3 Quality Control	44

3.4	Data Repair	45
4	Quality Assurance	46
4.1	Role in Keystone	46
4.2	Prototype Application	47
4.2.1	Software Specification	47
4.2.2	Software Design	51
4.2.3	Implementation Details	58
5	Quality Control	62
5.1	Role in Keystone	62
5.2	Requirements Elicitation and Specification	63
5.2.1	Requirements Elicitation	64
5.2.2	Functional & Non-functional Requirements	69
5.3	Software Design	73
5.3.1	High Level Design	73
5.3.2	Medium Level Design	76
5.4	Detailed Design	85
5.4.1	Server Side	85
5.4.2	Client Side	89
5.5	User Interface Design	95
5.5.1	Flags Interface	95
5.5.2	Data Visualization Interface	97
5.5.3	Administrative Interface	98
5.6	Implementation Details	101
5.6.1	Backend	101
5.6.2	User Interface	102
6	Data Repair	111
6.1	Role in Keystone	111
6.2	Implementation	111
6.2.1	Convex Optimization	112
6.2.2	GPU Adaptation	113
6.2.3	Further Optimization	114
6.2.4	Multi-GPU	116
6.2.5	Data Imputation	117
7	Validation and Discussion	118
7.1	Clustering Experiments	118
7.1.1	Experimental Setup	119
7.1.2	Data Imputation	119
7.1.3	Timings	119
7.2	QA Discussion	124
7.2.1	NRDC Impact	124
7.2.2	Broader Impact	126
7.3	QC Case Study & Discussion	127
7.3.1	Discussion	128
7.4	Validation of the Keystone Model	131
7.4.1	QA and QC	131
7.4.2	QC and Data Repair	132
8	Conclusions and Future Work	133
8.1	Conclusions	133
8.2	Future Work	134
8.2.1	Additional Data Imputation Algorithms	135
8.2.2	Ontology-Based Microservice & Database Generation	135

8.2.3	Modifications to the NRDC Schema	135
8.2.4	Source Agnostic Data Streaming with Kafka	136
Bibliography		137

List of Tables

2.1	An excerpt of time-series environmental data downloaded from the NRDC. The values depicted here are temperature measurements collected over the course of 7 minutes.	17
4.1	Functional requirements for the NRDC Quality Assurance Application.	48
5.1	The core functional requirements of the NRAQC Application.	70
5.2	The secondary functional requirements of the NRAQC Application. . .	71
5.3	The tertiary functional requirements of the NRAQC Application. . .	72
7.1	An excerpt of environmental data used for bench marking the effectiveness of the improved RSFKM clustering algorithm. The values depicted here are humidity and temperature measurements collected from many sites across Nevada.	118

List of Figures

1.1	An example of a cave painting from Lascaux, France. Photo taken by Mike Beauregard [11]. From a certain perspective, cave paintings can be thought of as some of the first data recorded by humans.	1
1.2	An in-situ environmental monitoring station in the Snake Range. Stations like this are hard to monitor regularly and can produce errors in data when damage is incurred to one of the many delicate sensors around the site. Photo by Scotty Strachan.	3
2.1	An example of geospatial data from NOAA showing recorded high temperatures across the United States, for a particular day. Taken from [22]	16
2.2	An example a chart graphing time-series meteorological data over time taken from [20]. Because of it's naturally linear nature, time series data is most commonly visualized using a line chart, although it sometimes is rendered with a scatter plot with no interconnecting lines.	18
2.3	CUDA Architecture for use with GPU programming, from [76]. The CUDA architecture abstracts the physical cores used for processing on an NVIDIA GPU into groups of threads which represent a single stream of execution. Each of these threads can be presumed to run in parallel to one another. These threads are then broken into blocks which are further generalized into grids. The right hand side of this figure shows the memory hierarchy of a CUDA-equipped GPU.	31
2.4	Architecture comparison between traditional Virtual Machines and containers, from [80]. In this diagram the difference between containers and virtual machines are highlighted from the hardware layer up. Containers are much more lightweight and share resources from the host operating system. Accordingly, it's much easier spin up more of them to perform smaller tasks than dedicated VMs.	33
2.5	An graph visualization of a description of a person using RDF syntax. The node on the far left is the subject of all of the nodes to the right. Each node is connected via an arc which represents the predicate which connects them.	35
2.6	The Ionic logo.	36
2.7	The Python logo.	37
2.8	The Flask logo.	38
2.9	The Angular logo.	39
3.1	The Keystone Model of data management. The three key processes are closely interrelated and together produce a complete data product.	42
4.1	Diagram showing use cases and their actor interactions for the Quality Assurance Application	49
4.2	An example of inputting a picture from the phone's camera into metadata for a particular system in the NRDC.	52
4.3	High-level block diagram detailing the architecture of the QA app. Client and server are connected via http calls from the mobile application to the Edge Micro services.	54

4.4	A mock up showing the design of the flag dashboard (left) and the data visualization dashboard (right).	55
4.5	Screenshots indicating the final UI design of the Quality Assurance Application. On the left is an updated and ascetically pleasing main menu screen. On the right, we see an example of a sublist in the hierarchical navigation structure. Here we see that Site Monitoring and Meteorological are systems associated with the Rockland Summit Site. Clicking on either item will populate a new list of Deployments that are in the chosen system.	57
4.6	An example of retrieving latitude and longitude coordinates from the phones GPS.	59
4.7	An example data entry page containing info about a single data sensor. The floating action button in the lower right-hand corner provides the option to add a service entry when clicked.	60
5.1	Website map for the NRDC Quality Control Application.	73
5.2	Component diagram showing high level architecture for server side of NRDC Quality Control Application.	74
5.3	Comprehensive Class Diagram for Quality Control Application.	77
5.4	The Configuration section of the Class Diagram.	78
5.5	The Datasource section of the Class Diagram.	81
5.6	The Test Suite section of the Class Diagram.	83
5.7	The Edge Classes section of the Class Diagram.	84
5.8	General Quality Control Activity Chart.	86
5.9	Activity Chart for Configuring the Microservices.	87
5.10	Activity Chart for retrieving the Measurements from the NRDC system.	88
5.11	Activity Chart for Monitoring any changes to the Configurations.	90
5.12	Activity Chart for Testing and Flagging of incoming Measurement data.	91
5.13	Activity Chart for Creating the Tests.	92
5.14	Activity Chart for Visualizing the Flagged Data.	94
5.15	Main landing page for the Quality Control Application. It is the flagged data dashboard and it shows at a glance the number of flagged measurements in need of expert review. Other pages can be navigated to via the horizonatal navigation bar on top.	95
5.16	Main landing page for NRDC QC Application. Note the changed data in the center of the doughnut chart as the cursor hovers over an arc representing a subset of flagged measurements.	96
5.17	Flags of a certain type organized by datastream. Either of these flag bundles, represented by blue circles, are active links to the data visualization page where the associated datastream will be automatically visualized.	97
5.18	The main data visualization page. A toolbar on the left hand side of the screen provided various functionality to the user. They can add a datastream to the visualization, manually flag datapoints in the graph by slicing areas, or get a detailed list of all flagged datapoints associated with this stream. Flagged data is colored red in the visualization.	98
5.19	Data visualization page showing interactivity of graph. Even non-flagged data can be viewed by hovering in this manner.	99
5.20	The configure tests screen. The user can navigate to a datastream to configure existing tests via the “view tests” button or they can construct an entirely new test by selecting “New Test”.	100
5.21	The new test creation page. Users are guided in creating a new test in a streamlined and step-by-step process.	100
5.22	The main landing page of the NRAQC application.	103

5.23	A page which organizes flagged data points by the stream they were flagged on. On this page we can see that there are 365 missing value flags, with 360 originating from one datastream and 5 from another.	104
5.24	An image of the main data visualization page for the NRAQC software. This datastream shows the results of inserting empty spaces to represent gaps in a dataset.	104
5.25	A list of all flags associated with the currently visualized datastream, with their timestamps and the name of the flag.	105
5.26	The top level options for configuration. User and data source configuration were not implemented for this prototype.	106
5.27	A snapshot of the hierarchical navigation used to find a particular datastream which can be tested. The image on the right was uploaded from the Quality Assurance Application detailed in Chapter 4. On the left, the nav history pane shows breadcrumbs of where we have come from. We can click on any of those breadcrumbs to return to a prior page.	106
5.28	A snapshot showing the variety of metadata we can retrieve from the QA app and display in the QC app.	107
5.29	A snapshot showing our options for creating new tests on a chosen datastream. On this datastream we already have an existing test for repeat values we can modify and we also can create a new test which checks for values which exceed desired ranges of our measurement series.	108
5.30	An example of a test configuration page. This particular test checks to verify that a datapoint does not exceed either the minimum or maximum range specified in these form fields.	108
5.31	When information has been input into a form field the save changes button appears.	109
5.32	When changes are saved the user is notified via a green strip at the bottom of this form.	110
5.33	The Demo page of the NRAQC app. This page notifies users about the current status of the demo via helpful text updates.	110
6.1	A diagram showing a basic breakdown of the implementation and operation of the core solver implemented as a CUDA struct from modified C code. This diagram shows the mapping of the solver to each block and indicates how the problem is decomposed, where each line in our membership matrix is solved in parallel on the GPU. The decision to call this solver kernel on one thread per block came from slowdown observed with more than one thread per block.	115
7.1	A line plot showing the runtimes of the RSFKM algorithm running sequentially on a CPU and in parallel on one, two, four and eight GPUs. The runtimes are per iteration of the clustering algorithm and averaged over ten trials. The dotted line indicates a trendline which shows the projected runtimes of a single GPU experiment. The equation next to the trendline was used to derive runtime values for efficiency calculations.	120
7.2	A line plot showing the per-iteration speedup factor of singular and multi-GPU experiments run with the RSFKM algorithm. For larger numbers of clustered values, the addition of more GPUs produces near equivalently scaled speedup factors.	121

- 7.3 A line plot showing per-iteration throughput of the RSFKM algorithm as run on CPU, one, two, four and eight GPUs. Throughput for this paper was expressed in terms of vectors/ms and was calculated as $thp = n/ms$ where n is the number of vectors input for clustering. GPU throughput handily outstrips sequential throughput by a wide amount, even when processing a small number of values. 122
- 7.4 A line plot showing the relative efficiency of multi gpu implementations vs. single GPU implementation. Calculated as $E = t_{gpu}/(t_{gpuN} * N) * 100$, where N is the number of GPUs being used, this graph indicates how much meaningful work each GPU is performing. 123

Glossary

in-situ environmental sensing A type of sensor-based research where a measurement device is located in the area it is observing. For example, a wind-speed monitor on a weather station is a in-situ sensing device.

data imputation The process of using statistical methods to fill a missing value in a dataset by leveraging known data which may be related to the missing point in some way.

interoperable A characteristic of data that expresses an ease of use across different programs and models.

machine readable A characteristic of data that indicates that it can be parsed by computers and computer programs.

metadata Data that describes data. In the earth sciences it is often used for data indexing in addition to ascertaining the provenance of data.

ontology A graph describing relations of ideas and words in a particular domain.

QA Abbreviation for “Quality Assurance.” The process of monitoring and maintaining the hardware and software which provides data. This process involves technical maintenance of hardware and software in addition to the documentation of metadata related to this process and detailing specific processes.

QA/QC An abbreviation for the closely linked processes of Quality Assurance and Quality Control which are part of data management.

QC Abbreviation for “Quality Control.” Describes the process of checking collected data for consistency and correctness. Used to ensure that collected measurements reflect ground truth values.

remote sensing A type of sensor-based research where a measurement device is removed from the area it is observing. For example, satellite imagery is a form of remote sensing.

sensor network A collection of sensor-based research sites capable of transmitting data between one-another.

time-series A type of data that measures a single variable over time. Mathematically represented as an vector indexed at timestamps.

vocabulary Similar to ontology. A lexicon of words which express ideas in a specific domain. In the sciences we want ensure that our vocabularies are as standard as possible.

1 Introduction

1.1 The Humanity of data

As long as 40,000 years ago, humans were recording their experiences on the walls of cave dwellings in the form of paintings discovered in modern-day Indonesia [6]. An example of such a painting can be seen in Figure 1.1. Although the purpose of such paintings is yet to be definitively established it cannot be denied that there is something undeniably human about them and at the same time something undeniably human about data.



Figure 1.1: An example of a cave painting from Lascaux, France. Photo taken by Mike Beauregard [11]. From a certain perspective, cave paintings can be thought of as some of the first data recorded by humans.

Long before we as a species knew what existed beyond the next visible horizon, or before we learned what forces moved the wind, the earth, the sun and the stars, we

made indelible, intelligible marks that recorded information. Whether intentional or not, this information conveyed something to humans living 40,000 years in the future.

Most fundamental definitions of data describe it as “facts or statistics” collected for analysis. Although these recordings could have been put to walls for pure aesthetic value, there seems to be some fact underlying these recordings. An ineffable and relatively insignificant data point which recounted some reality of early humanity. A data point which says, “I experienced something. Here it is.”

1.2 The importance of data

Today, thousands of years removed from these early “data points,” we are living in the midst of a data explosion. We are collecting more data, about more subjects, faster than we ever have before. In the last two years alone, we have generated over 90% of all data ever produced [61]. In this current “Information Age,” our entire existence is generating data.

Everything from our social interactions to our physical health is being recorded, mined, aggregated and analyzed. Never before has data been so important and so ubiquitous as it is today. Furthermore, this influx of data has transformed many domains in the sciences. For example, in the year of this thesis’ writing, over 5 petabytes of data were aggregated and collected to “take” the first ever picture of black hole [1]. This was so much data that it had to be shipped *by plane* to the location where it was being processed. This methods of transfer was significantly faster than transferring it over the internet.

In the earth sciences, in particular, this transformation has been no less impactful. Orbital satellites are capable of collecting millions of data points in a single second in the form of multi-spectral images. Even for the relatively less data-dense subdomain of *in-situ* environmental sensing, the advent of IoT and new approaches to data streaming has made continuous measurements of remote locations commonplace [65, 68]. Figure 1.2 presents an example of an in-situ environmental sensing site.

In a practical sense this means that we are collecting data in the earth and

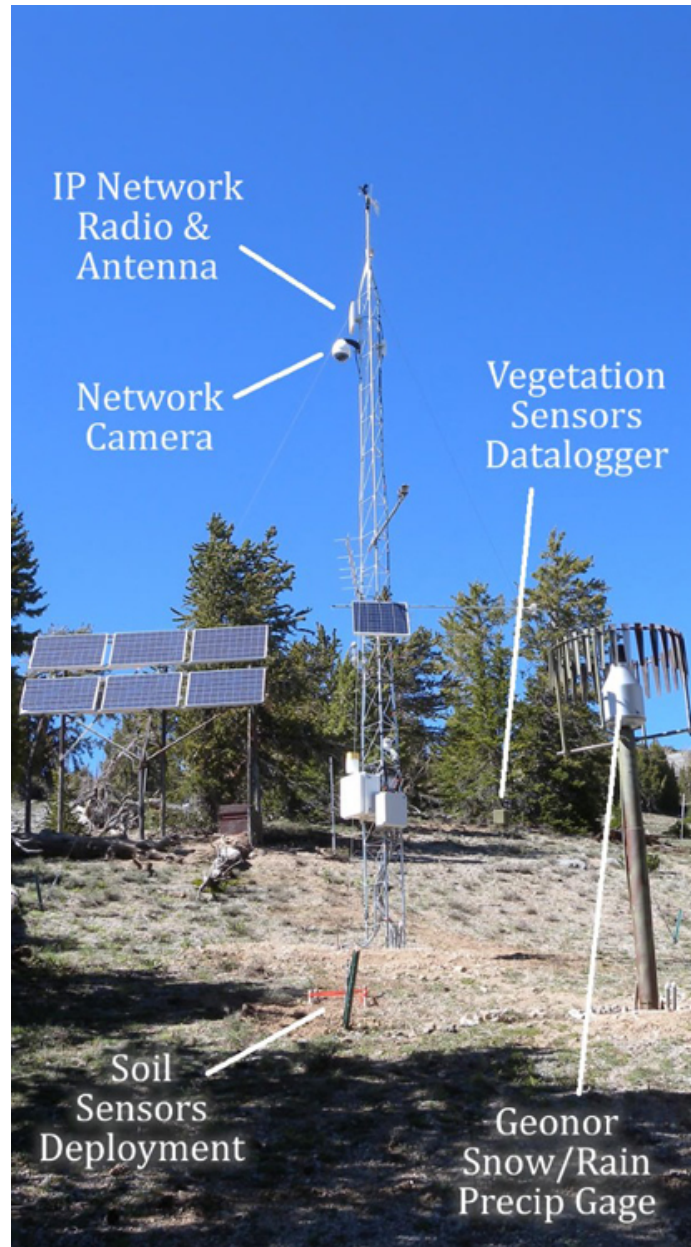


Figure 1.2: An in-situ environmental monitoring station in the Snake Range. Stations like this are hard to monitor regularly and can produce errors in data when damage is incurred to one of the many delicate sensors around the site. Photo by Scotty Strachan.

environmental sciences at volumes and velocities which we are unprepared for. More specifically, traditional approaches to metadata management and quality control are lacking [21]. Metadata has been traditionally handled with a combination of pen-

and-paper, excel spreadsheets and web interfaces for manual upload to a database if one existed. For quality control, many data sets are uploaded and downloaded from data repositories with no guarantee that any quality checks have been undertaken on them.

With a small volumes of data, these traditional approaches pose little problem when making a single data product for scientific analysis. Individual scientists can afford to spend the time manually uploading and binding metadata to their datasets. For quality control, individual scientists can write their own programs and routines to perform quality checks on static datasets they download. But this is still a problem in aggregate [108].

As datasets grow, the demand for more real-time processing grows with them to ensure that less data is laying fallow as raw data points and can be used to provide actionable intelligence. Accordingly, these traditional, manual approaches are no longer viable. In order to accommodate this new era of data collection, automation is required for metadata management, quality control and data repair to make data products at the same speed we are making data [21].

1.3 The Purpose of This Document

To address these contemporary issues facing the development of quality data products this thesis proposes the “Keystone” model of data management for streaming sensor data. This model suggests that three automated software modules working in tandem provide the necessary and sufficient functionality to take raw data and make a quality data product which can be used for sound science. Specifically, by automating Quality Assurance, Quality Control and Data Repair processes, data products can be created at a rate that keeps pace with the production of data itself.

As proof of this model, this thesis details the implementation and validation of three distinct software prototypes which derive mutual benefit from one another. First, a cross platform Quality Assurance mobile application was developed which makes the digitization and binding of metadata to existing data in the Nevada Re-

search Data Center nearly instantaneous. Next, a Quality Control application called the Near Real-Time Quality Control Application (NRAQC) was developed which leveraged the metadata digitized by the QA application. This application provided significant functionality for the automated testing of NRDC data. Finally, an imputation algorithm was modified to run on multiple GPUs to provide fast and accurate data repair on datasets with missing information.

In Chapter 2 we will expand on prior work for these software solutions in addition to identifying gaps in contemporary data management which drives the need for this thesis. We will then propose and specify our solution to this identified technology gap in Chapter 3 with the introduction of the Keystone Model. In Chapters 4, 5, and 6 we will outline, in detail, three prototype applications which have been used to fulfill the requirements outlined by our Keystone Model. From there, in Chapter 7 we will present a validation of the three prototype applications and the Keystone Model itself. Finally, we will conclude this thesis in Chapter 8 with a review of what was covered across these chapters and will propose several avenues for future work.

.

2 Background & Related Work

In this chapter we hope to introduce you to both the domain to which the proposed solution applies in addition to the gap it seeks to bridge. The content contained here should be sufficient to provide a surface level understanding of the terminology, tools and issues which affect data management in the earth sciences at the time of writing. If a particular idea is not explored in sufficient detail it is the hope of the author that you will find in this chapter a resource which will provide you with more detailed information or the terminology necessary to effectively research it on your own. In the era of Google, research can be as simple or as difficult as the vocabulary of a problem makes it.

2.1 Data Management

Data management broadly refers to the information science subdomain which is concerned with the acquisition, transfer, storage, maintenance and dissemination of data. The purpose of such practices are to ensure data collected at high volumes and velocities will be equally usable 10 years from now as it will be on the day of its collection. While this sounds like trivial problem to those with a passing familiarity of scientific data analysis it can severely inhibit scientific progress and the development of actionable intelligence [73].

To illustrate, imagine that you are a new environmental scientist graduate student. You just received a spreadsheet from your lab leader containing meteorological data and were told to analyze them to see if you can draw any conclusions about global warming in rural China. Now, immediately you should have some questions

about the metadata for these data. First, where were they collected? By whom? What devices were used to collect them? Were the data collected with the intent that they would be analyzed in this way? Were they collected by a trained scientist or an amateur? Did they do anything to ensure the data were accurate?

A week or two after data was collected these questions are easy to answer, especially if the project's primary investigator (PI) is in your department. If he didn't clearly document these things on paper he can give you answers to most of these details off the top of his head. You can jot them down and begin your analysis, radiant in your confidence that data management is a waste of time.

Now imagine that this data was collected not two weeks ago, but 20 years ago and not in the United States but instead in South Vietnam. You are uncertain if the data producer is still employed (or was ever employed) in the department from which this data originates, can communicate with you in English or is even still alive. Without proper data management the answers to most of these questions posed above turn from definite detailed descriptions to logical conjecture and pure guesswork. This dataset may have produced some invaluable, and never before published, data which answers your research questions perfectly, however without proper data management ensuring that contextual metadata is preserved you will be unable to use it. It will be worth less than the measurements you could make with a Mercury thermometer outside of your office door.

This problem is one which the domain of data management hopes to resolve by streamlining the processes of collecting, annotating, exposing and delivering data. While many projects and papers have explored how best to accomplish this goal, the most seminal comes from a work published in 2016: "The FAIR Guiding Principles for scientific data management and stewardship [108]."

2.1.1 FAIR Data Principles

In this highly influential work by various members of the FORCE 11 organization [43], the authors suggest four key principals which should govern Scientific Data Manage-

ment with the goal of promoting the discovery and utility of scientific data. These four principals are **Findability**, **Accessibility**, **Interoperability** and **Re-usability**. A significant portion of this work was developed with these principles in mind, so we will explain them in detail.

For a dataset or data to be considered *Findable*, it must meet several criteria. First it must be assigned a “globally unique and persistent identifier.” Oftentimes this identifier takes the form of a DOI, or digital object identifier [29]. Second, data must be “described with rich metadata.” This basically means that there must be detailed and abundant data which describes the data. This also requires that the data-describing-the-data must be closely tied to the data so the two are easily associated and aligned with one another. Finally, this principle requires that data is registered or indexed in a searchable resource. As datasets are often very large and sometimes nebulous its crucial that people have some means to only get those datasets which they require for their science. For earth scientists, this search-ability often takes the form of geospatial parameters, like location, with options to narrow data sets with additional information, such as time-of-collection and data-type [24, 71].

Moving into *Accessibility*, the Fair Data Principles dictate that a dataset can be called accessible if it meets two requirements: first, that data is retrievable by it’s identifier using a standardized communications protocol. As addendums to this requirement the authors indicate that such a protocol should be “open, free and universally implementable” [108] in addition to allowing for authorization and authentication procedures where necessary. Basically, the authors are suggesting that data should be available over the internet and accessible through HTTP[41] calls or, failing that, TCP/IP calls. Second, metadata must be accessible even when the data is no longer available. This requirement of *Accesability* is quintessential to a core goal of this thesis. It suggests an elevation of metadata to a position of key importance, far removed from its current status as an afterthought of data dissemination. This goal of collecting and managing metadata in a accessible and machine readable fashion

was the key impetus of the Quality Assurance software component of this thesis.

Following from *Accessibility* is the requirement that datasets be *Interoperable*. Specifically this means that data must: (1) “use a formal, accessible, share and broadly applicable language for knowledge representation”; (2) “use vocabularies that follow FAIR Principles” and (3) “include qualified references to other data/meta-data.” These three requirements can be synthesized to illustrate the idea that data should be easily usable by anyone who picks it up. Per the first requirement, it should be encoded in a standard format, like XML[18] or JSON[23]. These languages meet the requirements of interoperability because there exists significant support for them in a variety of commonly used languages. If we are using C# or Python we should be able read data with the same ease for each language. The next two criteria are a bit more abstract.

A vocabulary in this sense, is a technical term referring to a formalized lexicon of words used in a specific domain. More specifically in this context it means the terminology used to markup and organize my data. Take the following code snippet as an example:

Listing 2.1: An example of a vocabulary in the environmental science domain.

```
<data>
  <measurement>
    <unit>Fahrenheit</unit>
    <value>10.2</value>
  </measurement>
</data>
```

In this snippet, the language is XML. The vocabulary describes the terms “data”, “measurement”, “unit”, and “value.” Oftentimes to ensure that a vocabulary is FAIR it will be encoded in a formalized dataset itself as an ontology[3] and made accessible via a special type of identifier called a Uniform Resource Identifier (URI)[62]. The ontology itself guarantees FAIRness by using standardized vocabularies[31, 48] which can be referenced and accessed in a FAIR way. Although seemingly infinitely recursive, pretty quickly one arrives at a document which references itself and breaks the chain of referencing new documents.

Finally, for a dataset to be *Reusable*, it must meet one core criteria: it (and its metadata) must be “richly described with a plurality of accurate and relevant attributes.” [108] To clarify, Wilkinson [108] specifies some crucial attributes which should be included. Specifically data should be released with: a clear data usage licence and detailed provenance. Additionally, data should meet community standards. In short, data should be released as a complete product with information about where it came from, what’s inside it, and how to use it. It should also meet the standards that a given community upholds as a quality product.

2.1.2 Data Management in the Earth Sciences

With FAIR data practices we have identified a gold standard of Data Management which can be applied to evaluate the quality of data management in nearly any field. For the earth Sciences, FAIR data practices have been widely discussed and are being adopted as a policy by numerous institutions including the American Geophysical Union [92]. However, for many intuitions, there exist barriers which hinder the easy adoption of FAIR data management practices. Relevant to this work we will explore two currently existing in the earth sciences: workflow barriers and technological barriers.

Workflow Barriers

Workflow barriers describes the complications which arise from traditional data management practices in the earth sciences. There has been significant scholarship on this topic in the last decade, as data management education, especially in the earth sciences, has failed to keep pace with the speed and volume of collected data [97]. According to Strasser and Hampton[97], “few courses exist that are exclusively devoted to data management practices.” They go on to assert that – even when data management practices are taught – spreadsheets are the most common tool used for data collection and education.

According to Jones this ubiquitous use of spreadsheets can be problematic for

effective data management due precisely to the factors that make them excellent tools for individual data analysis [53]. Their extensibility, UI-based formatting and organization, and ability to embed processing and charts make spreadsheets an obvious choice for analyzing data with minimal overhead. However, these same characteristics cause problems when attempting to disseminate the data contained in these spreadsheets. Spreadsheet software often uses proprietary functions for mathematical operations and markup of text/data. If a spreadsheet is the sole repository of collected data for a project, formatting and processing information needs to be stripped away from the core data before it can be usable. Furthermore metadata is often not well encoded or well associated with specific data values and is often stored as the headers of columns in spreadsheets. Additionally, vocabularies used in these headers are rarely described in a machine readable and interoperable format and are not accessible anywhere than in the dataset itself.

This illustrates the idea of workflow barriers by showing us a traditional data management workflow in the earth sciences. For a small project, data will be collected, aggregated into spreadsheets, organized with some identifying metadata and quality controlled for analysis. Metadata describing data provenance – including Quality Assurance practices performed on data collection instruments – will be poorly tied to datasets themselves: written up on pen-and-paper notebooks or as comments in embedded systems code [81, 87]. This workflow — although useful for answering specific research hypotheses — does not lend itself to reproducibility. Data managed in this way is oftentimes single-use; even if it is uploaded to a publicly accessible data discovery platform.

Technological Barriers

Following from the aforementioned Workflow Barriers compromising adoption of FAIR data standards there are also significant technological barriers which prevent adoption of these standards. Chief among these problems are limited capabilities and experience on the part of domain scientists in the area of software engineering. Although

most (if not all) are familiar with some programming languages (R, Matlab, and Python) and some libraries needed for building and analyzing environmental models, few domain scientists tend to use these languages to their fullest capabilities [63, 79, 85, 101]. Like data produced and used in one science project, many programs built by environmental scientists are single-use with the goal of reading, analyzing and outputting their data. Accordingly, this makes the adoption, use and implementation of general software solutions for common data management problems very difficult.

Although there have been significant efforts made by organizations like NEON[10], Earth Science Information Partners[35], EarthCube[33] and the Long-Term Ecological Research Network[50] to use and disseminate standardized software packages for data management, there exists significant diversity in hardware and software used for managing data in a FAIR way.[108] This means that although good solutions for Quality Assurance and Quality Control exist, their usefulness varies depending on the software stack or workflow utilizing them[34]. (These existing solutions will be explored in more detail in Section 2.5.) The use of these software packages are also limited nonetheless by the technological literacy of their users and their integration into real time pipelines becomes increasingly reduced as data ingestion workflows become sufficiently complex.

2.2 Earth Science Information Partners

The Federation of Earth Science Information Partners (ESIP) is a key organization in the domain of data management for the earth and environmental sciences. Comprised of member organizations like NASA[70], NOAA[75], and USGS[104], it's stated mission is to "To support the networking and data dissemination needs of our members and the global earth science data community by linking the functional sectors of observation, research, application, education and use of earth science." [36]

Accordingly, ESIP is not a standards or research organization but rather a platform for collaboration and exchange of information. This makes it invaluable resource to new researchers attempting to survey the domain of data management in environ-

mental science. The organization of ESIP is extremely sympathetic to the introduction of new members, being organized into informal clusters which hold monthly meetings for the sole purpose of exchanging new information relevant to their sub-domains.

Although there may be multiple clusters which are of interest to a computer science student or researcher in general, the most important to the topic of this thesis is the Envirosensing Cluster. The wiki page dedicated to this cluster's work provides innumerable resources which compliment the topic of this thesis, especially QA/QC [37].

2.3 The NRDC

The Nevada Research Data Center (NRDC) is the primary data management platform which the tools in this thesis were built for [77]. It currently hosts approximately 3.5bn time series data points collected from sites across Nevada and over 10m images collected daily from webcams mounted at those same research sites.

2.3.1 History

Originally built as a data portal/repository for data collected as part of a Track 1 NSF EPSCoR funded project, the NRDC began life as the Nevada Climate Change Portal(NCCP) [72]. As further funding was acquired through a second Track 1 Proposal for the Solar Energy Nexus Project, the NCCP was transformed from a data repository scoped to the house the data from one project, to a multi-project repository and data management platform for earth science research in Nevada and renamed the Nevada Data Research Center.

2.3.2 Position as a Data Management Platform

Although sustaining considerable growth since its inception years ago, the Nevada Research Data Center remains a relatively small platform for the dissemination and management of data in the earth sciences. Compared to many national platforms like NASA's EarthData [70] or CUAHSI's Hydroshare [24] the NRDC occupies a

very niche space as a data management platform. Although it hosts billions of data points, they are produced from a limited number of projects contained almost entirely within the state Nevada [72, 96, 102].

Although this sounds like a mark against the NRDC this limited focus is very helpful to the development and maintenance of the NRDC as an integrated platform. Unlike these other, larger data management platforms, the NRDC does not need to aggregate data from myriad unknown sources. Furthermore, it is able to focus its management practices on time series data alone, as opposed to geo-spatial data. This can simplify many workflows and make integrated tool sets much easier to build and configure to work with NRDC data. Furthermore, demands for space and processing tend to be more limited as the variety of data managed by the NRDC stays confined to this geographical area and this domain.

2.4 Earth Science Data

“Earth science” describes a very broad umbrella of loosely related fields which study the earth. It aims to model and explain the natural phenomena occurring across various portions of this blue marble we call home, from the reaches of the upper atmosphere (Meteorology) down to the depths of the earth’s core (Geology) and across our vast bodies of water (Hydrology) [54]. In order to study these qualities of earth, and especially those in the critical zone of life, significant data must be collected about as many natural processes as possible. Everything from wind speed, to soil conductivity, to precipitation and solar radiation must be collected and managed in some way. Although measurements can be extremely varied in their purposes and methods of collection, in the earth and environmental sciences they can largely be broken into two large subcategories: geospatial Data and time series data. These two categories are defined primarily by their method of collection and what they are used for.

2.4.1 Geospatial Data

Although none of the solutions proposed in this document operate on or are configured to work with geospatial data, I have included an overview of this type of data because it is ubiquitous in the earth and environmental sciences. Furthermore, although there exists a need for Quality Assurance and Quality Control software that works on data like this, solutions are being actively developed by large organizations like ESRI [38, 39].

Informally, geospatial data can be thought of as any data we can overlay on a map. In Figure 2.1 we see a variety of geospatial data that everyone is familiar with: a weather map displaying temperature gradients over the united states. If you have ever been to weather.com you have seen a map like this. The data itself is geographically indexed temperature values at a single point in time rendered as colored pixels on this map.

Although meteorological data is very commonly stored and rendered in this way, it is not the only data which can be called geospatial. Some other types of data which are commonly managed geospatially are: hydrological data[105], geological data[5], and even data from the social sciences, like political maps used to organize political affiliations by location [93].

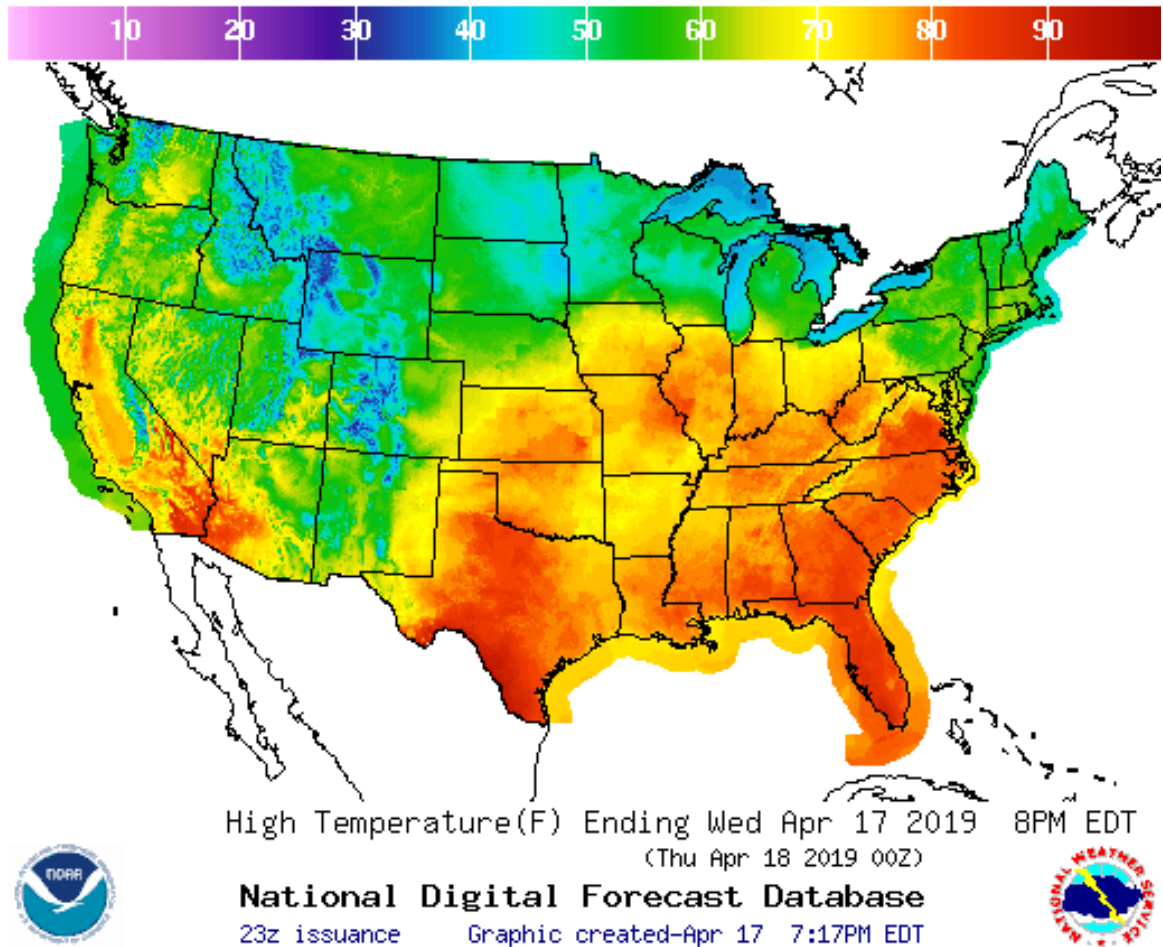


Figure 2.1: An example of geospatial data from NOAA showing recorded high temperatures across the United States, for a particular day. Taken from [22]

2.4.2 Time Series Data

By contrast to our consideration of geospatial data in Section 2.4.1, an understanding of time series data is crucial to understanding the purpose and implementation of the solutions proposed in this work. Time series data is defined mathematically by David Billinger in a textbook on time series methods as a vector of values s.t. X is a set of real numbers and $X_j(t)$, $j = 1 \dots r$ where “ r is real valued,” and t increments or decrements by equidistant steps. The author goes on to qualify this purely mathematical definition by saying that t often is a timestamp of some variety.[20]. An example of time series data can be found in Figure 2.2 and Table 2.1.

Table 2.1: An excerpt of time-series environmental data downloaded from the NRDC. The values depicted here are temperature measurements collected over the course of 7 minutes.

Timestamp	Temperature (C)
1/25/2018 1:19	-5.678
1/25/2018 1:20	-5.654
1/25/2018 1:21	-5.697
1/25/2018 1:22	-5.774
1/25/2018 1:23	-5.788
1/25/2018 1:24	-5.793
1/25/2018 1:25	-5.797

In more practical terms, this just means that a time series is a vector of numbers indexed at timestamps. Or even more concretely, in the scope of this paper, its a measurement of a single variable collected at set intervals over time. Time series datasets, measuring one value over time, are commonly identified as having some other characteristics which make them unique. First, points located near one another in time are expected to be more closely related or interrelated than measurements farther removed from one another. Two, being linear with only two features, they allow for the use of mathematical regression to build models to predict future events. Finally, as they are typically less dense than geospatial data, they use up significantly less space in storage and less processing power when being manipulated.

By comparison to it's geospatial cousin, time-series data does not measure a wide area or hope to provide information about a very large area. It quantifies the conditions in a very small area with a fine granularity. In the environmental sciences, we use time series data to glean a very deep understanding of a point in space. This data can obviously be used to make conclusions about similar climates and natural phenomena, however the research questions associated with the collection of time-series data are often very specific, with broader implications. For example, "What can the age of trees in the Great Basin tell us about forest fires? [14]"

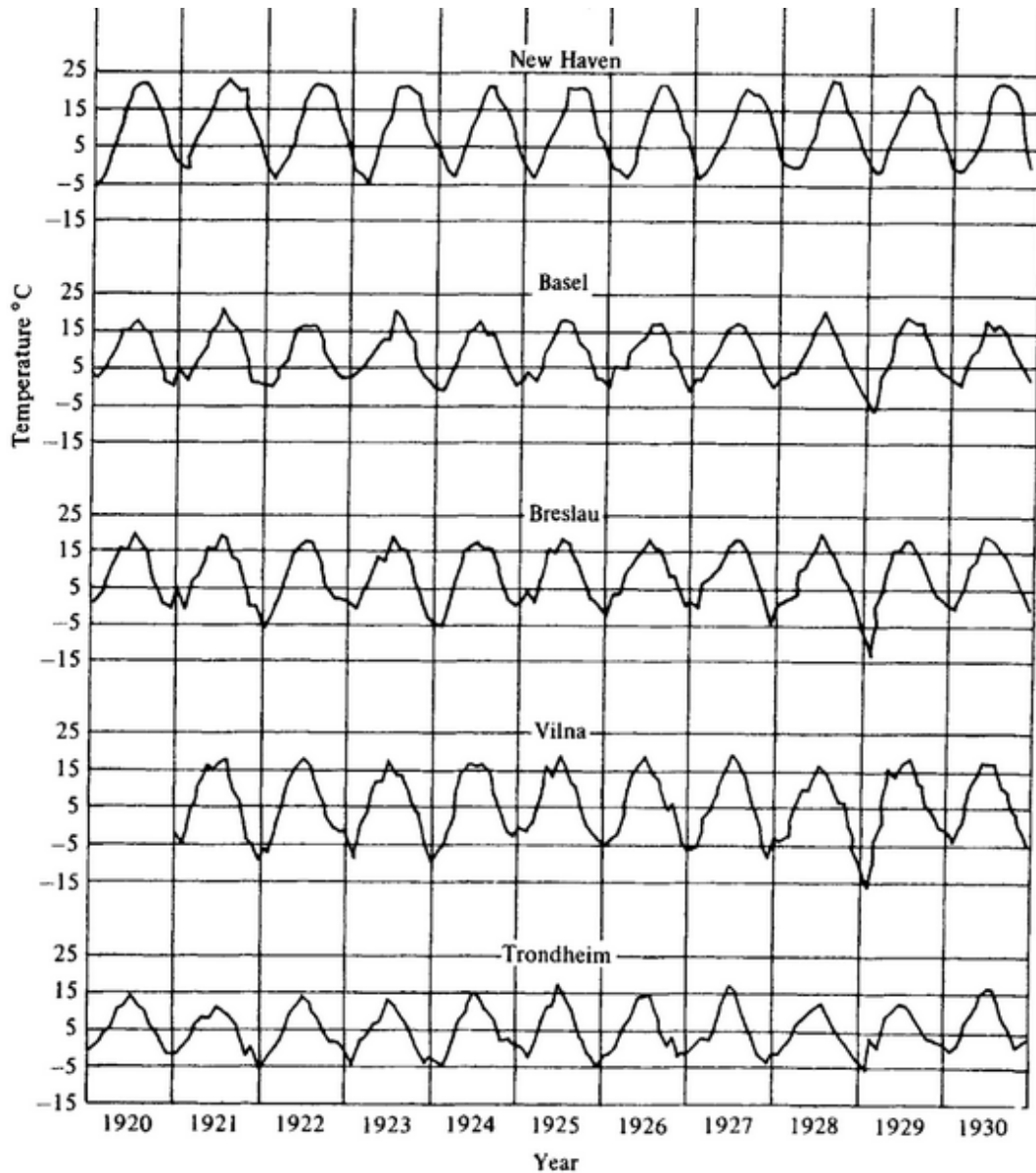


Figure 2.2: An example a chart graphing time-series meteorological data over time taken from [20]. Because of it's naturally linear nature, time series data is most commonly visualized using a line chart, although it sometimes is rendered with a scatter plot with no interconnecting lines.

2.5 Quality Assurance and Quality Control

Chief among problems currently in need of resolution for data management in the earth sciences are the twin issues of Data Quality Assurance and Data Quality Con-

trol. As the volumes and velocities of data collection increases, there is an increasing need to automate both of these processes to the greatest extent possible. In the following subsections I will explore these problems in greater detail and outline some of the extant software solutions which have been proposed to address these issues.

2.5.1 Quality Assurance

Individual researchers and one-off projects dominate the model of data collection in traditional climate/environmental research [65]. In traditional research, single use data proves extremely effective at answering a singular projects research questions and fulfilling research requirements attached to funding streams. However, despite the short-term success of such a model, a clear problem arises when another research team wishes to use this previously collected data, or when data need to be integrated into larger syntheses [15]. This problem drives the need for complete, accurate, and usable metadata.

Metadata is considered a major part of the data life cycle. Creation of metadata include information surrounding the data set, such as format, file names, and measurement units, and information about the experiment producing the dataset, such as documenting data processing steps and contextual information to the data [67]. Its purpose is to make the datasets easy to understand [67]. Unfortunately, that is not always the case. Traditionally, scientific metadata is kept in notebooks and papers, a holdover from days before computers. This creates fractured data, where its hard to relate the electronic data sheets to hand written documents [97].

Due to the narrow focus of typical projects, only the original researchers intimately know how the data was generated. Metadata is often non-standardized, incomplete, and stored in temporary formats. Some communities and organizations have developed their own metadata standards; however, several different metadata standards can exist within any given discipline [46]. Eventually, over time – or given enough distance – the value of these data sets is diminished to other researchers and the public. It becomes harder to recover and ascertain contextual information that

is essential to decoding it and metadata standards rarely address long-term preservation [100]. Methods for uniform quality assurance and metadata collection are being recognized as the next major challenge for data intensive science as collection becomes increasingly automated and results globally disseminated [108].

At the broadest level, Quality Assurance refers to the preventive maintenance and management process employed to reduce inaccuracies in data automatically logged by sensors [34]. Although many works touch on the idea of Quality Assurance, the most seminal motivating work published on the subject comes from a 2013 paper “Quantity is Nothing without Quality: Automated QA/QC for Streaming Environmental Sensor Data” [21]. In this paper, the authors put forth a comprehensive, generalized set of practices to optimize QA on environmental field sensors. They suggest that QA procedures be automated, well documented, and complete metadata maintained alongside data. This work presents Quality Assurance as a process oriented approach to data management, this strongly implies that no single software solution can provide effective QA but can only rather aid the QA process performed by humans. Accordingly, a significant number of background works on this subject study the analysis and creation of software that best facilitates good QA practices.

A further example of a motivating work indicating the need for Quality Assurance practices in the paper “Workflows and extensions to the Kepler scientific workflow system to support environmental sensor data access and analysis” [9]. This paper outlines the development of a software environment that addresses technical challenges related to accessing and using heterogeneous sensor data from within the Kepler scientific workflow system. Within the bounds of their end-to-end examination of this existing sensor network workflow, the authors indicate on several occasions that a clear need exists for quality assurance practices with in-situ sensor networks. They also acknowledge that existing software solutions used to facilitate these practices are not well developed. Our QA application intends to fill this proposed gap.

Outside of motivating works driving research in /glsQA, there also exists several works that explore the practical or theoretical implementation of Quality Assurance

processes and software. One, Meta-information concepts for ecological data management represents an early survey of many data management needs for ecological sensor data collection [66]. This paper suggests exactly at which stage in the data collection pipeline to place QA processes and software. Another paper, “Anomaly detection in streaming environmental sensor data: A data-driven modeling approach” shows a more practical approach to Quality Assurance by suggesting that expected shifts in the quality of data can be anticipated by using data-driven modeling techniques. Although the approach taken by the authors of this paper is not representative of the approach we undertook, it shows that there exists a strong interest in applying modern software engineering techniques to the problem of Quality Assurance.

Along similar lines, in the paper, “Automatic processing, quality assurance and serving of real-time weather data” the authors demonstrate that there exists a strong interest in developing software to automate and streamline the process of Quality Assurance on environmental sensor data [109]. The authors of this paper propose a software to manage and utilize statistical metadata that can indicate the quality of data through uncertainty values. The concept of collecting metadata in a standardized format for quality assurance strongly reflects the goal of the Quality Assurance software developed and detailed in this paper for the NRDC. However, our approach is unique in that it is oriented towards metadata collection as an end goal rather than as a supplement to Quality Control computations. Taken together, the above papers generally indicate a strong research interest external to the Nevada Research Data Center (NRDC) in the development of Quality Assurance software, however there also exists a well-documented interest in developing software within this organization as well.

References to the considerations of a Quality Assurance system for the NRDC appear in early literature proposing practices and architecture for its predecessor project NCCP [65]. These works present Quality Assurance and Quality Control (QC) as crucial elements of any large scale environmental research project. They also impress upon the reader a need for a standardized and centralized set of tools which

enable universal comprehension of data being collected. From this specific need to improve on existing QA practices, a quality assurance application was conceived.

2.5.2 Quality Control

In the field of in-situ environmental sensing, invalid data is a very serious concern that affects the overall quality of a data set and pollutes the integrity of any derived data product. Invalid data are those measurements that do not actually represent the conditions they are supposed to represent. A hardware failure can often cause this divergence in representation and reality and produces gaps in data or verifiable incorrect values. Invalidity, in the context of time-series environmental data can be organized into five major categories: missing values, out of bounds, repeated values, spatially inconsistent values, and internally inconsistent values.

Missing values represent a period of time in which no measurements were logged. The out of bounds category details a measurement that exceeds maximum or minimum bounds of what is considered a reasonable measurement. Repeated values occur when the exact same value is logged in direct succession for an extended period of time. Spatially inconsistent data describes co-located physical data loggers recording vastly different trends in data over the same period of time. Finally, internally inconsistent data describes logical inconsistencies within a measured data stream or between two related datastreams.

The problem of Quality Control (QC) in the realm of environmental sensor data has been explored extensively in scientific literature. In work as early as 1970, techniques to formalize Quality Control have been detailed as a necessary part of any project which produces large volumes of autonomously collected data [40]. In more recent years however, as the autonomous environmental research sites have become cheaper to install, easier to set up, and generally more ubiquitous, there has been a strong surge of research exploring the methodologies and importance of Quality Control on high volume streaming data.

Many academic and industry oriented white papers published in the last 15

years explore the importance of Quality Control as it applies to domain specific data. Meteorological data, environmental data, metadata studies: together these and many more articles paint a clear picture indicating that Quality Control is no small problem to the scientific community working with time series sensor data [42, 49, 66, 108]. Outside of these general considerations however some key literature has examined major ideas informing modern automated quality control in great detail.

At the forefront of seminal works on this issue – and the one most informative to the software produced for this paper – is the article by Campbell [21]. This paper presents a clear and critical overview of Quality Assurance(QA) and Quality Control in the context of streaming environmental sensor data. As environmental data collection becomes increasingly automated, a need to build automated checks on these processes arises. As the paper indicates, Quality Assurance is “*process oriented*” and Quality Control is “*product oriented*”, establishing a clear delineation between the two often muddled concepts. From these basic definitions, the authors expand on automated QA/QC providing a survey of state of the art in this domain: best practices, implementation suggestions, and examples of prominent extant software which successfully automates these processes.

Significant work has been done by several organizations to create robust and practical QC software. Principal among these is a software suite called the GCE Toolbox by Georgia Coastal Ecosystems LTER. This Matlab based toolbox [59] represents the gold standard of QA/QC software for environmental sensor data. Users of this software can perform automated quality control manually or with rules-based checks, visualize data and import/export data in various inter-operable and standardized formats. Unfortunately, an advanced knowledge of Matlab—and access to the costly Matlab compiler and IDE—is required to use these tools. This greatly restricts accessibility to quality QC, especially to smaller research organizations with fewer resources. The Near Real Time Quality Control Application (NRAQC) addresses this drawback with its simplicity of deployment and multiple robust interfaces for configuration.

Outside of the GCE toolbox, several other software packages attempt to address

contemporary QC needs in various scientific domains. Chief among these, ArcGIS provides functionality for the automated QC of geospatial data [38]. Additionally, Campbell Scientific has support software associated with their data loggers called Loggernet. This proprietary software provides basic Quality Control functionality on streaming data [84]. NRAQC is not actively iterating on any functionality provided by ArcGIS as it only handles Time-Series data. Additionally, since NRAQC queries a database, file, or API for logged data, it is agnostic about hardware setups unlike Loggernet which interfaces directly with the physical data loggers, causing compatibility issues.

In addition to the software development efforts, many organizations are working to standardize the metadata flags which indicate specifically how a measurement was deemed invalid. Chief among these organizations are ESIP and QARTOD, with the former cataloging and supporting present efforts towards standardization and the latter putting forth a strong proposal for flagging standards in oceanographic sensor data that can easily be applied more widely [35, 47]. Standardization is crucial for data inter-operability. Understanding these standardization efforts is equally necessary to create QC software that scientific organizations will use.

The approach of utilizing a microservice architecture in a data intensive system is a recent development in distributed architectures. Even more, utilizing a microservice architecture to implement any form of quality control system is currently not a popular topic of research. It is because of these reasons that there is a severe deficit of academic papers within the last five years pertaining to both quality control and microservice architectures. With this said, this thesis provides both a solution to the problem of uncoordinated quality control, and a contribution to the research regarding usages of service-oriented architectures in quality control.

Finally, the primary drive and motivation to produce this software follows from the development of a Quality Assurance application that manages metadata for the NRDC [87]. Quality Control solutions represent a natural extension of metadata management practices. Data quality flags are essentially granular metadata bundles

applied to individual measurements, which denote any possible errors with the data, the means by which it was flagged and possible corrections made to the data. Together QA and QC represent two pillars supporting robust and usable data in the 21st century.

2.6 Data Imputation

“Missing data imputation”, or “gap filling,” describes the process of filling in holes that occur in sufficiently large data sets [94]. To ensure valid statistical analyses with a dataset these holes must be filled with accurate estimations of “ground truth” values. Although many simple and effective statistical imputation methods exist, like K Nearest Neighbors (KNN) which takes a simple mean of “closely related” data points [13], not every dataset is as sympathetic to these approaches as others. Accordingly, when choosing approaches to data imputation, multiple factors must be considered. The randomness of missing data and the structure of the dataset being imputed are a few examples of this.

2.6.1 Unsupervised Machine Learning

When classified, machine learning breaks down into two large sub-areas which each describe a collection of algorithms that share a common approach to learning: supervised and unsupervised[95]. With supervised learning, an AI is “taught” how to solve a particular problem by letting it:

1. Attempt to solve a problem on it’s own, blind.
2. Evaluate how close or how far it was from solving this problem.
3. Modify the tools or paths it uses to solve the problem according to how close or far it was from solving it.

This is an extremely non-technical overview of supervised learning but an in depth understanding of supervised learning is not crucial to understand this thesis.

But, it is important to understand how it differs from unsupervised learning. Now, the key idea of supervised learning is that the “correct” answer to the problem we are attempting to solve is known and can be compared against the attempt made by the AI. For example, if we ask a supervised machine learning algorithm to attempt to fill in a missing data value in our dataset we can compare it’s final guess against the real value and ask it to try again if it is too far from the actual answer [60].

With unsupervised machine learning the approach is fundamentally different, rather than having a correct answer to compare the AI’s attempts against we let the algorithm organize itself into the best groups it can find with no external feedback on whether they are correct or not. This does not mean, however, that an unsupervised learning algorithm does not receive any feedback or use any metrics to learn, it just doesn’t compare its guesses against the ground truth.

By comparison to supervised learning, unsupervised learning using objective metrics to evaluate if two data points (or likely data vectors) are similar or not. In KNN, mentioned above, and K Means Clustering the metric is the same: a measurement of distance between vectors. The assumption with this is that if two vectors which are closer in the n-dimensional space they occupy will be more similar than vectors which are farther away from one another. With KNN, every vector is compared directly against their closest neighbors to and can give a very clear idea of how far any vector is away from any of the other vectors in a dataset. With clustering, the main algorithm used in this paper for data imputation, all vectors in a dataset are compared against a set number of “centroids” each of which define the center-point of a particular cluster.

By organizing data into clear groups we can then fill in missing values by finding out which group a vector fits into best even if it’s missing one value compared to all others. Once we find it’s group we can then make an educated guess about what the remaining data value could be.

2.6.2 Clustering and Convex Optimization

The primary algorithm used for clustering in this thesis is called Fuzzy K-Means (FKM). Fuzzy K-Means clustering algorithms are modifications of traditional K-Means clustering. These algorithms exploit fuzzy set theory to define membership as a percentage allowing for a data point to have membership in multiple clusters [32]. This approach to clustering has proven to be very effective for applications in the domains of computer vision and pattern recognition [8, 106]. Data imputation as an application of FKM has been explored by many authors [57, 58], but in the recent past, “pure” FKM approaches to Data Imputation have fallen out of vogue, yielding to hybrid approaches with other machine learning techniques [7, 98]. Accordingly, there currently exists a research gap in the area of utilizing a standalone FKM algorithm for Data Imputation.

The traditional approach to Fuzzy K-Means, as introduced by Dunn[32], is relatively simple. A set \mathbf{X} of n objects can be grouped into c clusters with membership coefficients \mathbf{U} defined by centroids in matrix \mathbf{V} with the following algorithm:

Algorithm 2.1 Traditional fuzzy k means algorithm.

```

while Not Converge do
  Compute centroids  $\mathbf{V}$  via (Eq. 2.1)
  Compute coefficients of memberships for  $\mathbf{U}$  via (Eq. 2.2)
end while

```

The equations referenced in Algorithm 2.6.2 follow here with a brief explanation:

$$v_k = \frac{\sum_x w_k(x)^m x}{\sum_x w_k(x)^m} \quad (2.1)$$

$$\sum_{i=1}^n \sum_{j=1}^c u_{ij} = \frac{1}{\sum_{k=1}^c \left(\frac{\|x_i - v_j\|}{\|x_i - v_k\|} \right)^{\frac{2}{m-1}}} \quad (2.2)$$

u_{ij} in the above equation describes a specific cell of membership matrix \mathbf{U} which defines the membership of object x_i in centroid v_j .

The fundamental algorithm modified and used in this work is derived from Xu [110]. This algorithm, called “Robust and Sparse Fuzzy K-Means Clustering,” makes several adjustments to traditional Fuzzy K Means Algorithms to enforce “robustness” and “sparsity” of the clustering. In this context, “robustness” refers to a characteristic of centroids which mitigates outlier influence in updating their mean value. “Sparsity” refers to a cluster’s membership characteristics. A membership vector is sparse if a data point is wholly a member of only one cluster and no others. It is the authors’ assertion that there exists an optimal sparsity for each data point where it belongs to a few clusters but not others.

The algorithm proposed by Xu modifies the core FKM algorithm expressed above with the modification of Equations (2.1) and (2.2). Several supplementary equations were also introduced by the authors of this paper to enforce robustness and sparsity. Equations 2.3 and 2.4 replace (2.2) and (2.1), respectively:

$$\min_{\substack{u^i=1, \\ u^i \geq 0}} \|u^i - \tilde{h}^i\|_2^2 \quad (2.3)$$

This equation optimizes membership for a value denoted by row u_i , where $1 \leq i \leq n$, and n is the number of rows in our data matrix. Equation (2.3) is a wholly independent sub problem for each line. So with n values this minimization can be performed entirely in parallel with n cores. \tilde{h}^i is an auxiliary variable used to enforce sparsity that is stored in an auxiliary \mathbf{H} matrix. The next equation updates the centroids and replaces (2.1):

$$v_k = \frac{\sum_{i=1}^n s_{ik} u_{ik} x_i}{\sum_{i=1}^n s_{ik} u_{ik}} \quad (2.4)$$

s_{ik} in this equation is an auxiliary variable which enforces robustness. If a euclidean distance between a centroid and data vector is within a user defined threshold then s_{ik} will be defined as the reciprocal of that distance, which reduces the weight of farther points. If its outside the threshold then s_{ik} will be defined as 0. This prevents outliers from influencing centroid updates. u_{ik} describes the membership percentage of data vector i in cluster k , and x_i describes data vector i out of n values being clus-

tered. As many calculations in this algorithm require numerous auxiliary matrices and sum reductions plentiful opportunities for parallelization could be found.

Constrained optimization was mentioned many times in this subsection and deserves a quick overview. If you remember Calculus I, optimization is a mathematical method which will give us the x value we can input to get the smallest or largest y given a function $y = f(x)$. With constrained optimization this problem becomes complicated because we now put some constraint on the x value we want to know, eg. $x < 1$ and $x > 0$. Now we are looking for the smallest or largest value of x that fulfills this condition as well. This becomes increasingly more complicated as x becomes an n -dimensional vector [17]. Fortunately, we do not require a deep understanding of how to solve these problems on paper as certain software tools already exist which solve these problems for us. Most important at this point is to have an idea of what is meant by the term “constrained optimization”.

2.6.3 Data Imputation Methods

The work of Schmitt surveys data imputation methods in detail with a comparative analysis of six common methods of data imputation [83]. In this paper, the authors compare the imputation methods of a simple Mean, KNN, fuzzy K-means, singular value decomposition (SVD), Bayesian principal component analysis (bPCA) and multiple imputations by chained equations (MICE). Using a quantitative analysis gauging the accuracy of imputed data on multiple benchmark data sets, the authors largely concluded that – for both large and small datasets – FKM provided accurate results at the cost of a very poor execution time. This trade-off indicates a clear opportunity for optimization with the aid of modern GPU programming techniques.

Data imputation with Fuzzy K-Means algorithms is relatively simplistic. After all data objects have been clustered, a missing data value j for a specific data vector

x_i can be filled in with the following equation [57]:

$$x_{i,j} = \sum_{k=1}^V U(x_i, v_k) * v_{k,j} \quad (2.5)$$

To explain this formula in greater detail, $U(x_i, v_k)$ describes a membership value for a particular vector x_i in a cluster v_k . This value between, 0 and 1, can be used as a weight to scale how much a given centroid k should influence the sum which yields our missing value. The sum itself is the summation of the value at feature j in each centroid. With the weights, this should accurately place our missing value between all centroids of which it has some membership, this should also accurately reflect it's true value.

2.7 GPU Programming

In this work Graphical Processing Units (GPUs) are leveraged to address the issue of Fuzzy K Mean's relatively slow execution times compared to other imputation methods. Accordingly discussion of GPU programming will primarily be constrained to a general overview and a background of its use for these applications.

In 2007 NVIDIA released its propriety programming language CUDA which enabled programmers to leverage the thousands of cores in GPUs for high-density parallel processing. In Figure 2.3 you can see a visual breakdown of the abstracted architecture the CUDA programming language uses to simplify the complexity of massively parallel programming on GPUs [74].

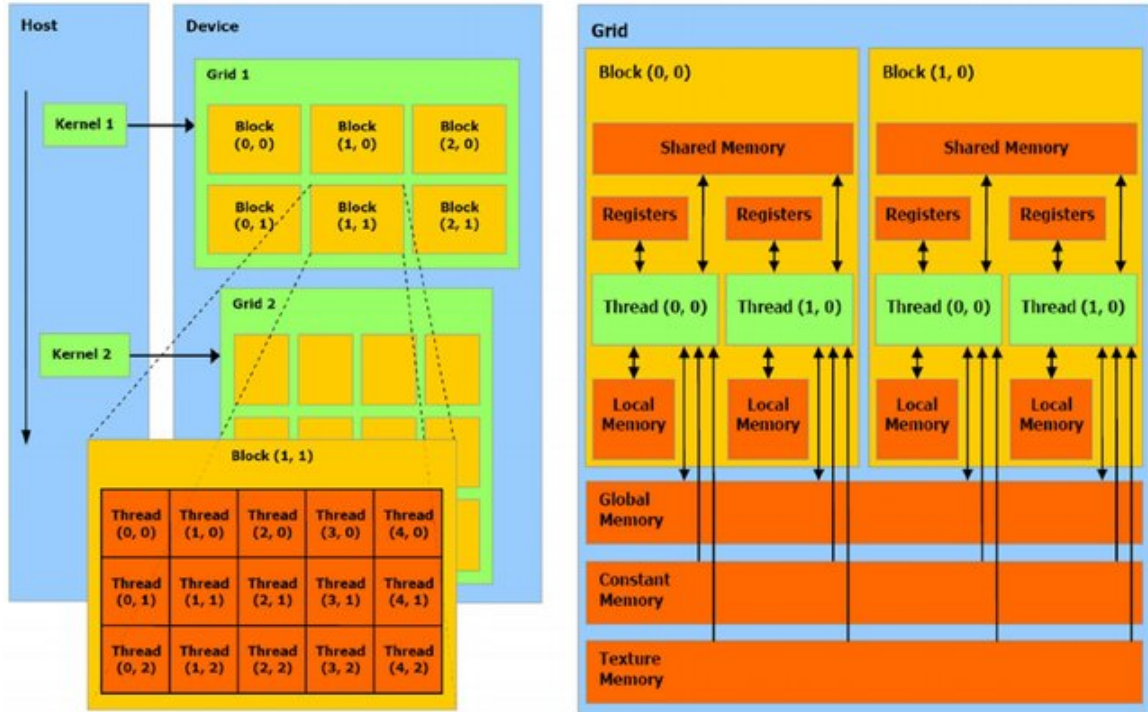


Figure 2.3: CUDA Architecture for use with GPU programming, from [76]. The CUDA architecture abstracts the physical cores used for processing on an NVIDIA GPU into groups of threads which represent a single stream of execution. Each of these threads can be presumed to run in parallel to one another. These threads are then broken into blocks which are further generalized into grids. The right hand side of this figure shows the memory hierarchy of a CUDA-equipped GPU.

To describe Figure 2.3, in greater detail, on the far left we see a box called a “kernel”. A kernel is basically the name for a CUDA subroutine that runs on a GPU. Any given kernel should strongly resemble a C program because CUDA uses C syntax for nearly the entirety of grammar. When a kernel is executed it is assigned, from the calling program on the CPU, a number of Grids, Blocks and Threads. This organizational hierarchy does not directly map to any real part of the GPU’s hardware; however, it abstracts the assignment of certain threads of execution to specific GPU cores away from programmers. With CUDA, programmers can treat each individual thread like its own contained stream of execution (it is very much like the more commonly found multi-threaded processing enabled by tools like pthreads). This or-

ganization of processing units allows for programmers to map their processing to the data they have, which is a significant change compared to CPU-based programming which requires programmers to map their data to the single processing stream they have available.

In addition to this abstraction of execution streams, CUDA also provides a clear memory hierarchy which separates out what data a given thread has access to at any given time. This memory hierarchy gives significantly more control to programmers so they do not need to worry about overwriting or otherwise messing up their large datasets while performing massively parallel operations.

The concept of leveraging GPUs to accelerate the time consuming process of clustering is not a new one. A handful of publications have been produced exploring how best GPUs can be leveraged for clustering. However, many of these publications are now outdated [90, 91], having been released very shortly after the introduction of NVIDIA's GPU processing framework, CUDA. Accordingly, they do not reflect significant changes to GPU hardware and software. Changes which certainly affect the structure of proposed algorithms and implementations. Additionally, more prominent contemporary articles do not consider applications of data imputation and, more importantly, do not create Multi-GPU implementations of their algorithms [2]. We assert that the usage of Multiple GPUs for this algorithm is a substantial contribution of this thesis over prior work.

2.8 Critical Tools and Technologies

In this final section, we will give a brief overview of some software tools and technologies which were used in the construction of the software solutions detailed in this thesis.

2.8.1 Containerization

Containerization refers to a minimalistic approach to virtual machines and roughly describes a recent trend in cloud computing. The core idea underlying containeriza-

tion is that, instead of spinning up full virtual machines, emulating the full hardware and software which can be found on a single server, it would be more processing and space efficient if we only emulated the small parts needed to one program or a handful of scripts. A general comparison of these two approaches to virtualization can be seen in Figure 2.4, which was drawn from an blog post comparing containers and virtual machines [80].

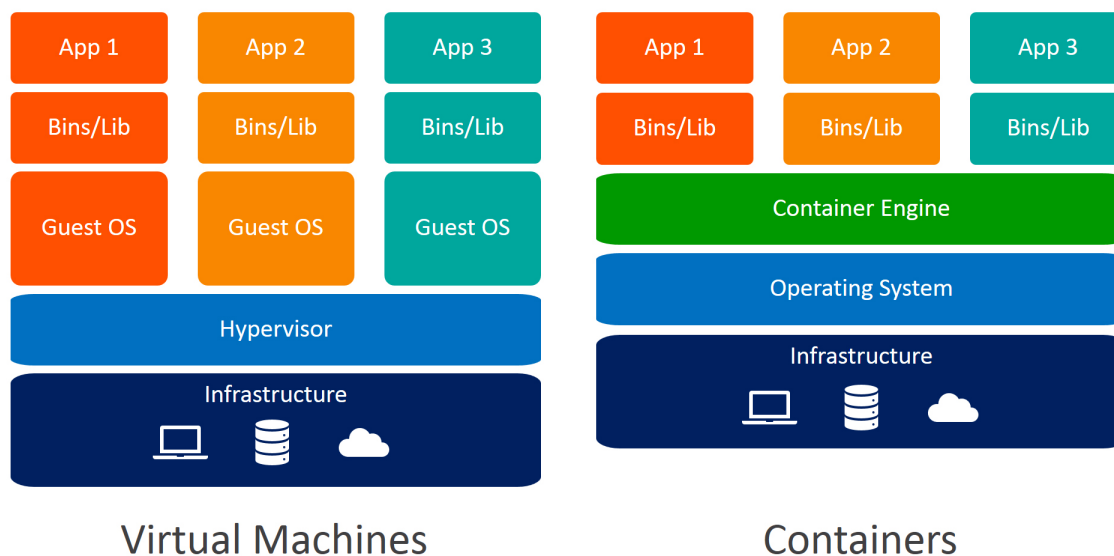


Figure 2.4: Architecture comparison between traditional Virtual Machines and containers, from [80]. In this diagram the difference between containers and virtual machines are highlighted from the hardware layer up. Containers are much more lightweight and share resources from the host operating system. Accordingly, it's much easier spin up more of them to perform smaller tasks than dedicated VMs.

Although there exist many varieties of container software, the one used most prominently in this thesis is Docker [27]. Docker is containerization software platform which provides: repositories for existing containers, tools for orchestrating multiple containers across one or multiple servers[28], and a scripting language which users can use to define their own custom containers. Its one of the most effective ways to make a software deployable anywhere as all possible dependencies can be easily hard coded into the container environment.

2.8.2 OWL & RDF

OWL and RDF are lexical standards short for Web Ontology Language and Resource Description Framework, respectively [3, 25]. RDF, originally developed to provide a machine-readable language for describing basic relations and information on the internet. Using a logical syntax of “triples,” where any two ideas and their relationship can be broken down into a subject, predicate and object. The following code snippet, drawn from the Friend of a Friend Specification[19], demonstrates how it works:

Listing 2.2: A small ontology which defines a person using the Friend of a Friend (FOAF) vocabulary.

```
<foaf:Person rdf:about="#danbri
                xmlns:foaf="http://xmlns.com/foaf/0.1/" >
  <foaf:name>Dan Brickley </foaf:name>
  <foaf:homepage rdf:resource="http://danbri.org/" />
  <foaf:openid rdf:resource="http://danbri.org/" />
  <foaf:img rdf:resource="/images/me.jpg" />
</foaf:Person>
```

This very basic snippet describes a person. Its comprised of 5 “triples” all concerned with one subject: “#danbri.” This fancy “#danbri” name is basically declaring a variable which exists uniquely in the namespace of our document and provides an absolute reference for all the following objects.

When visualized, an RDF description of an idea or ideas produces a Directed Graph. These graphs can be very complicated, or very simple but they follow a common structure where vertices represent subjects or objects and edges represent predicates. The graph for the above description can be seen in Figure 2.5.

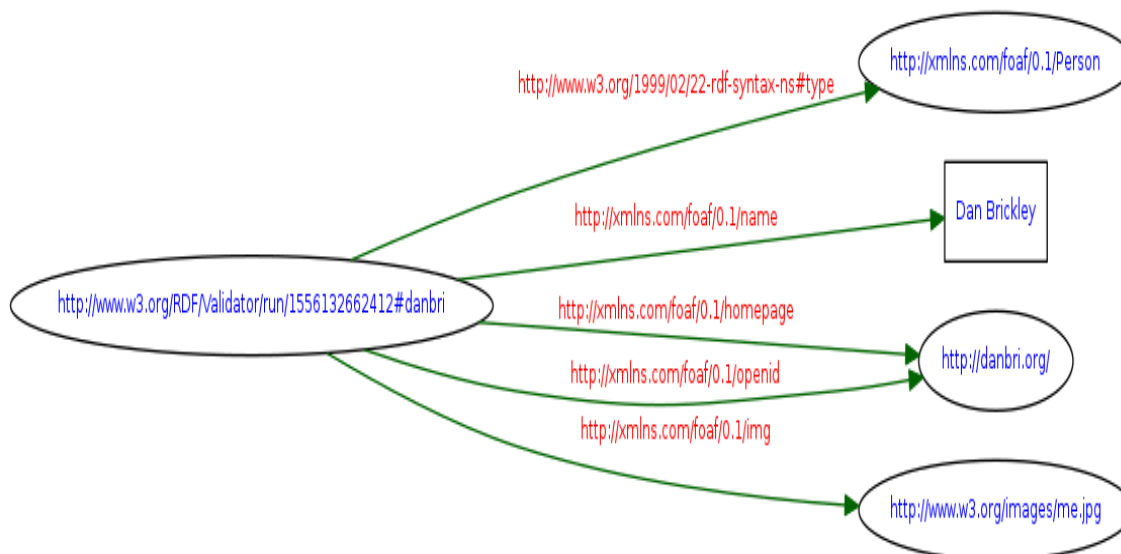


Figure 2.5: An graph visualization of a description of a person using RDF syntax. The node on the far left is the subject of all of the nodes to the right. Each node is connected via an arc which represents the predicate which connects them.

After working with RDF for a few years, information scientists found it to be lacking in semantic richness. It can describe relations between ideas but only in a very abstract way. Furthermore it does not allow for significant divergence from a one way logical association between ideas, limiting its ability to describe complex set functionality. To address this, researchers at the World Wide Web Consortium (W3C), defined a new specification called OWL.

OWL extends the definitions of RDF by allowing users to define their own predicates with significant detail. With OWL users can define a much more complex language in a machine readable format that captures the subtlety of ideas which are related but not in a direct sense. OWL enables users to define relations between ideas like membership, disjointedness, and intersectionality. This allows for a much richer description of objects and ideas on the web.

2.8.3 Ionic

Ionic is a mobile application development framework which uses JavaScript and the Angular framework, along with HTML and CSS to develop cross-platform native Android and iOS applications [52]. Currently on its third iteration, Ionic allows users to develop mobile, desktop and web applications which run as though they were native. Ionic was used to develop the Quality Assurance application detailed in this thesis. The logo of this software can be seen in Figure 2.6.



Figure 2.6: The Ionic logo.

2.8.4 Python

Python is a commonly found scripting language used to develop a wide variety of software products and applications. The Python logo can be seen in Figure 2.7. With a syntax which mimics natural language more than programming languages like C or C++, it is used significantly by computer scientists, software engineers and domain scientists alike [79]. Because it is interpreted, rather than compiled, and performs garbage collection, it is criticized for being less efficient than more technical languages like C++. Accordingly, Python is rarely used for hyper-performant applications like embedded systems programming.



Figure 2.7: The Python logo.

Nonetheless, for many applications Python makes it extremely easy to spin up a project and begin coding. There also exists significant support for many external libraries which provide functionality for everything from multi-threading applications to performing machine learning tasks. In Keystone, Python was used for the development of microservices in the Quality Control Application, the generation of code in the new development on the QA application and also for managing data and hosting kernels for the clustering application. To expand on any of the work detailed in this thesis and good grasp of Python will be necessary.

2.8.5 Flask

Flask is a Python framework which provides easy to use tools for the fast creation of microservices [82]. Installed via the Pip package manager, a few lines of code can set up an endpoint ready to be queried at localhost in seconds. All the microservices created for the QC portion of this thesis were written in Flask. The Flask logo can be seen in Figure 2.8.



Figure 2.8: The Flask logo.

2.8.6 Angular

Angular is a front end web framework developed by Google to simplify the development of complex and dynamic web applications [45]. The logo of this framework can be seen in Figure 2.9. In its original form it was called Angular.js and could be linked for use in a website via a simple `<script/>` HTML tag. After upgrading to 2.0, the Angular project dropped the “.js” from its name and moved from a pure JavaScript framework to a TypeScript framework. TypeScript, by comparison to its less restrictive cousin, is not interpreted but rather compiled and enforces strict typing on all its various language constructs [103]. This change made Angular 2.0 significantly different from Angular.js. The syntax, build and deployment methods changed significantly between these two versions.



Figure 2.9: The Angular logo.

The Quality Assurance software detailed in this thesis used an early version of Ionic which required Angular.js as the core language/framework of development. The Quality Control application built for this thesis, attempted to iterate on lessons learned from the Quality Assurance application by utilizing the new and improved version of Angular for its front end development: Angular 2.0. Accordingly, a familiarity with both Angular.js and Angular will be required to understand the code produced as part of this thesis.

2.8.7 CVXPY

CVXPY is a Python library which can be installed via the Pip Python package manager [26]. It provides an interface between any Python program and multiple varieties of convex optimization algorithms. It was used in this thesis to solve the constrained convex optimization required for a sequential implementation of the Robust and Sparse Fuzzy K-Means algorithm detailed in Chapter 6. The particular solver this library utilized for this problem was the ECOS solver [30].

2.8.8 CVXGEN

CVXGEN is a web service which generates library-free optimized C code for a particular optimization problem [64]. This library was used to generate C code, which was adapted into CUDA code, to provide a means to perform convex optimization in the GPU adaption of the Robust and Sparse Fuzzy K Means Algorithm.

2.8.9 PyCUDA

PyCUDA is a Python library which provides methods for interfacing with GPUs for massively parallel programming [78]. Capable of calling kernels, passing CUDA code to these kernels, and assigning threads and blocks at runtime, the PyCUDA library provides all functionality which standard C++ interfaces provide. This library was used in this thesis to enable the use of Python for reading in, preprocessing and managing data before it was passed to GPUs in the Improved Robust and Sparse Fuzzy K Means software.

3 Data Quality Model

In Chapter 2 we identified a few key problems plaguing modern Data Management practices for the earth sciences in general and some gaps which can be filled in the NRDC workflow, in particular. To enumerate them very clearly they are:

1. A need for FAIR data/metadata management practices to promote the usability of data products.
2. A need for automated and integrated quality control applications which can keep up with the increasing volume and velocity of data being collected in Nevada.
3. A need for fast and accurate data repair which can fill holes identified by solutions to gap 2.

In order address these three gaps I propose an integrated and related framework of applications which solves each of them in turn. When combined, these software provide a comprehensive solution to many data management requirements of the NRDC. Together these software packages make up the focus of this thesis, Keystone, and can be described with “The Keystone Model.”

3.1 The Keystone Model

The Keystone Model, shown in Figure 3.1, is the comprehensive solution proposed by thesis for the above three problems in modern data management for the NRDC and beyond. Comprised of three core modules, which represent software solutions

developed to fulfill each need above, this figure shows how they inter-operate and the key input and outputs of the Keystone toolset.

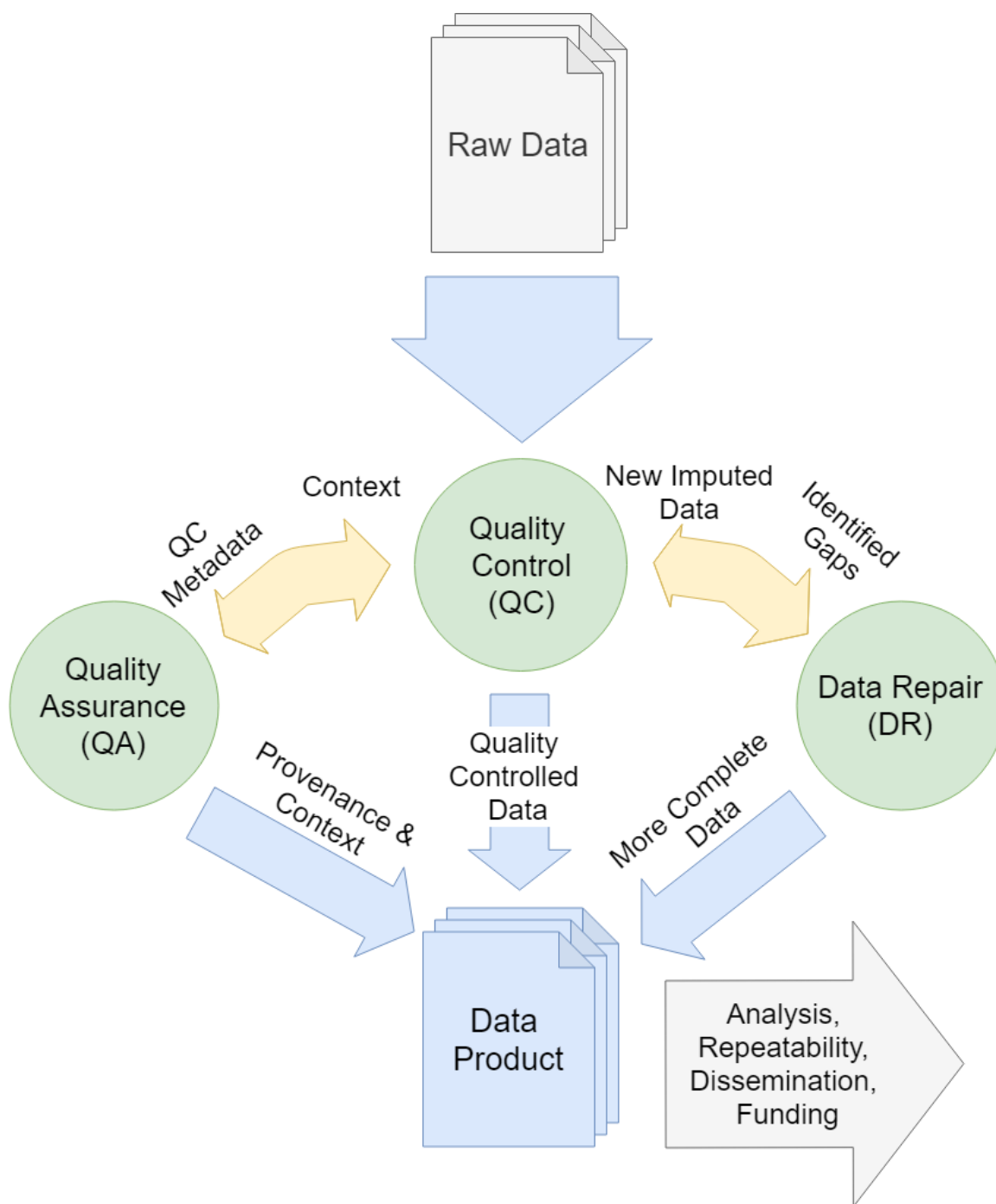


Figure 3.1: The Keystone Model of data management. The three key processes are closely interrelated and together produce a complete data product.

This model is very high level however and does not specify the specific applications developed for this thesis. Rather it identifies how an integrated tool set which hopes to provide comprehensive data management should be structured. Starting from the top, we see that the primary input into our model is raw data. This means it has undergone no quality control and does not have significant metadata bound to it. It's either a collection of files or perhaps a series of rows in a database table. Most importantly, it is most certainly not a data product and has very limited use, even to the individuals who collected it.

Moving to the bottom of this figure, we see the ultimate goal of Keystone and data management in general, a usable “data product.” By comparison to the raw data above, a data product is a usable package of data. It has been quality controlled, it has granular metadata clearly organizing the data, it has global metadata identifying the context surrounding its collection and the provenance of the data itself, including what measures were taken to ensure that the data was collected under the best circumstances possible. This final data product can be used in many different ways – by anyone, anywhere – for scientific analysis, for experiments which test scientific repeatability, to acquire or justify funding which can continue research on these topics.

This transformation from a pile of raw measurements to a complete data product is enabled only through the coordinated of Quality Assurance, Quality Control and Data Repair processes. These three processes are often distributed throughout the data collection and analysis phases of the overarching scientific process. This uneven and undefined distribution of processes has many drawbacks, which are explored extensively in Chapter 2.

If we link these processing steps as shown in Figure 3.1 and integrate them into a single system the process of producing a comprehensive data product becomes much more streamlined. Each of these processes can be automated to some degree which enables data management pipelines to keep up with the ever increasing volumes and velocities of streaming data coming in from an increasing variety of sensors and

instruments.

Accordingly, to demonstrate the effectiveness of this model, three software solutions have been developed that aid, or provide the functionality required for Quality Control, Quality Assurance and Data Repair. Respectively, the corresponding software solutions are the NRDC Quality Assurance Mobile Application[69, 88], the Near-Real Time Autonomous Quality Control Application (NRAQC)[86], and finally the Improved Robust and Sparse Fuzzy K Means (iRSFKM)[89].

3.2 Quality Assurance

As explained in Section 2.5.1, Quality Assurance is the least software-oriented process included in the Keystone Model. Nonetheless, it has significant opportunity for improvement and automation through the use of software. Identifying clear problems with traditional methods of metadata management which contributed to the first gap detailed at the beginning of this chapter, the Mobile Quality Assurance Application was developed to simplify the input and binding of project-level metadata which is used for quality assurance processes.

Further details on the purpose, design, implementation and outcome of this application can be found in Chapter 4.

3.3 Quality Control

Detailed in Section 2.5.2, Quality Control as a process allows for significant automation. However, there currently exists a lack of integrated and fully autonomous solutions to QC. To address this issue we have built the NRAQC system to provide integrated, quality control functionality to the NRDC. Although only a prototype it fulfills many requirements expected of such a software package.

Further details on the purpose, design, implementation and outcome of this application can be found in Chapter 5.

3.4 Data Repair

Data Repair is a non-technical term we have opted to leverage for use in this thesis. It merely refers to the act of filling in identified holes in datasets which have gone missing, for various reasons. Although commonly performed as part of the analytics process, by the scientist building a model or experimenting on collected data, an opportunity exists to expedite the process and make a more complete data product by performing this sort of repair after gaps have been found by autonomous quality control software. As indicated in Section 2.6.2, the main method used for data repair in this thesis was a clustering algorithm, modified to run on multiple GPUs.

Further details on the purpose, implementation and outcome of this application can be found in Chapter 6.

4 Quality Assurance

4.1 Role in Keystone

The Quality Assurance application detailed in this chapter fulfils the intended purpose of the Quality Assurance module in Keystone through its automated metadata creation, management and binding. The Quality Assurance Application produces machine readable metadata from the time of it's first entry and clearly binds that metadata to the data being collected from that site. Furthermore, because the structure of input metadata is enforced by form fields in the application interface, the vocabulary and organization of uploaded metadata is guaranteed.

These functionalities serve the overall Keystone model in several ways. First this metadata creation and storage provides information which is crucial to the transformation of raw data to a full data product. The metadata uploaded and stored in the NRDCs databases provide crucial contextual information needed to identify the source, types, and purpose of otherwise contextless measurements. The Quality Assurance application allows users to input information about what devices were used to collect data points, where they were located and the deployment conditions they were installed under. This information is crucial to understanding a data product when far removed from the source of its creation.

Furthermore, the Quality Assurance application provides functionality which binds crucial provenancial information to data products. By organizing and keeping track of service entries and documents associated with various tiers of NRDC site and sensor deployments, our final data product has a clear lineage of what processes were undertaken to ensure that collected data was accurate. Through the documents

which can be bound to various parts of this hierarchy, an idea of the original research questions which created this data can also be bound to data points in the form of associated papers and proposals.

Finally, the NRDC Quality Assurance application serves the next stage in our data management pipeline by providing machine readable metadata to our Quality Control software. This contextual information is similar to that bound into the final data product, however it can be used more dynamically to simplify workflows and provide richer interfaces to an integrated Quality Control platform.

4.2 Prototype Application

In order to prove the functionality expected of such an application that fulfils the requirements of the QA module, a prototype application was developed. The following subsections detail the specification, design and implementation of the NRDC QA application.

4.2.1 Software Specification

The QA App was developed with many functional and nonfunctional requirements in mind. These requirements were decided on after extensive talks with several data management experts and stakeholders. Detailed in Table 4.1, these requirements guided design and development of core functionalities for the QA application. Using an agile development method, these requirements went through several iterations before settling into the current list.

Using these functional requirements, a series of use cases were constructed and mapped in a use case diagram, found in Figure 4.1. This process informed the principal design phase of this applications construction and was frequently referenced or tweaked alongside the software specifications through the implementation phase.

Table 4.1: Functional requirements for the NRDC Quality Assurance Application.

Functional Requirements	Description
Input Data	The user shall be able to enter new data into the QA app.
Upload Data	The user shall be able to upload data to a secure database.
Read Data	The user shall be able to download entries from the database to their mobile device and view previous data entries.
Edit Data	The user shall be able to edit previous entries and upload the change data to the database.
Navigate Data	The user shall be able to move between different screens of the app, and input data.
User Authentication	The user will be able to authenticate themselves to access secure functionalities.
Delete Server Data	The user, with proper authorization levels, shall be able to delete data stored on the database.
Upload Photos	The user will be able to launch the device's camera and upload a photo from their photo gallery on their device.
Save Unsynced Data	The user shall be able to save data locally on their device to upload it to the server at a later point

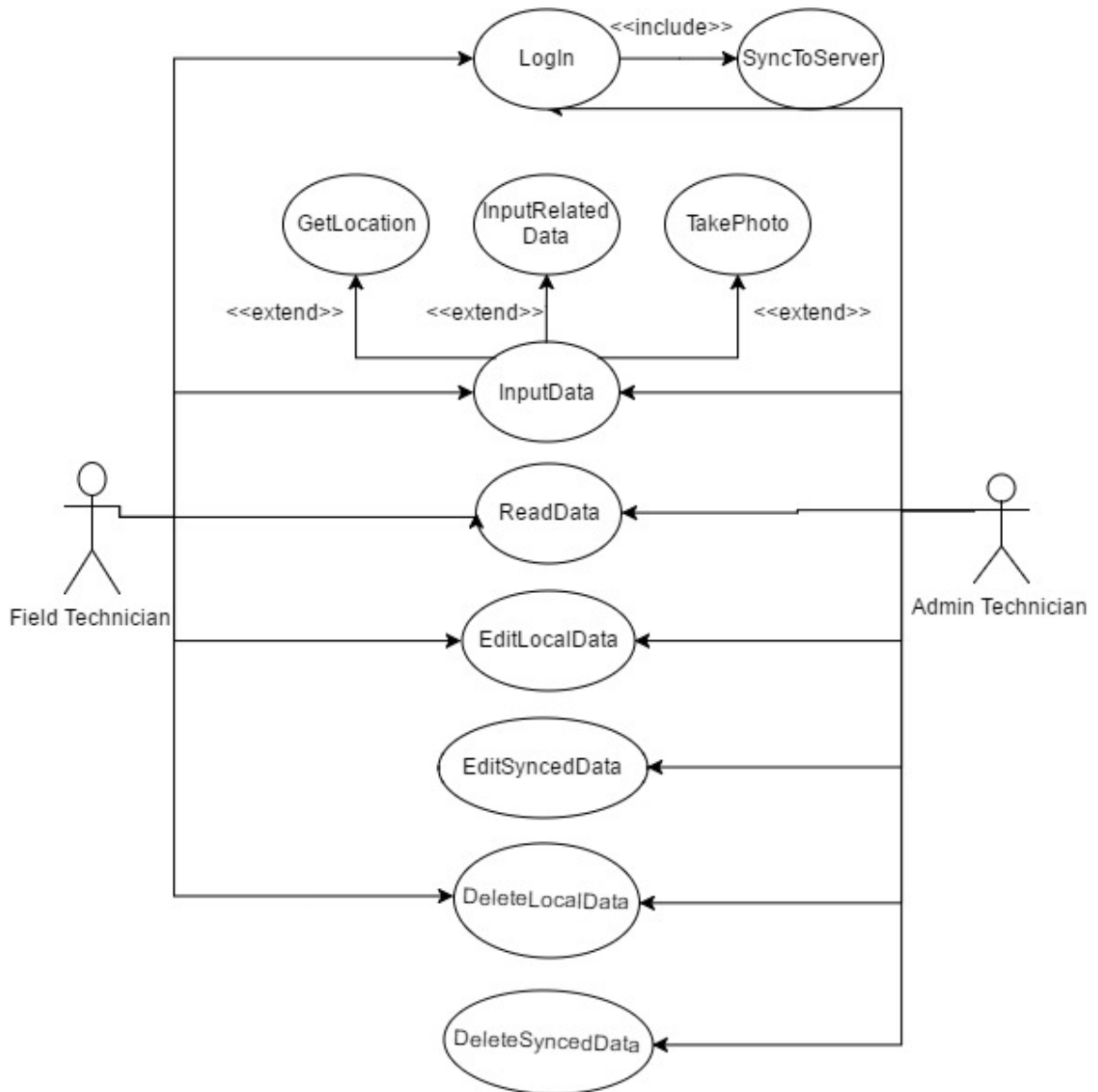


Figure 4.1: Diagram showing use cases and their actor interactions for the Quality Assurance Application

The description of each Use Case follows:

- LogIn

Field technicians must log into the app. Once logged in, technicians are given a list of projects they are associated with. This helps reduce the amount unnecessary data downloaded. Technicians can also add new entries and upload them to the server. Admin technicians, once verified through the log in, are given the ability to edit or delete entries already synced to the server.

- SyncToServer

Connects to the server and uploads new data entries found on the phone. Then, downloads new data found on the server. The app can manually be synced by pressing the synchronization icon on the header bar on the front page.

- InputData

Allows the user to input new data. Opens a blank template for whichever dataset they are choosing to input. Once finished, it is saved to local memory until synced to the server.

- TakePhoto

Opens the phone's camera app to take a picture that can be uploaded alongside the data set, like the System in Figure 4.2. Not every data set can have a photo.

- GetLocation

Uses the phone's GPS to fill out latitude and longitude coordinates, such as the Site in Figure 4.6. Only two types of data sets need GPS location.

- ReadData

All users must be able to view the data, regardless of whether or not they are logged in. This is so users who are not a part of the project, but are interested in the data, can view it. To read the data, users need only to navigate to their

desired object and click on the name.

- **EditLocalData**

Users are allowed to edit entries that have not yet been synced to the server. Users can navigate to unsynced data entries and select the edit button to change them.

- **EditSyncedData**

Admin technicians are allowed to edit data already synced to the server. If an admin is logged in, they can edit entries by navigating to it and clicking the “Edit” button. If the user is not an admin, this button will be greyed out. The changes will be uploaded the next time the app is synced to the server.

- **DeleteLocalData**

Users are allowed to delete entries that have not yet been synced to the server. Users can navigate to unsynced data entries and select the delete button to remove them.

- **DeleteSyncedData**

Admin technicians are allowed to delete data entries already on the server. If an admin is logged in, they can delete entries by navigating to it and clicking the “Delete” button. If the user is not an admin, this button will be greyed out. The changes will be uploaded the next time the app is synced to the server.

4.2.2 Software Design

When designing the Quality Assurance Application, there were several key requirements that shaped the development process. First, the QA application must be able to structure the enormous amounts of service entries and access them in a timely manner. To achieve this, the application utilizes a data access hierarchy that narrows down the amount of data queried. Second, the application must handle the situation that there is no available internet or cell signal in the immediate area. The application

The screenshot shows a mobile application interface for editing a 'System'. At the top is a dark blue header with a white back arrow and the text 'System'. Below the header, the form contains several fields: 'Power' with the value 'Negligible', 'Installation Location' with the value 'Sheep Range', and a dropdown menu currently showing 'Scotty Strachan'. At the bottom of the form, there is a photo gallery area with a small thumbnail of a landscape and a blue button with a white camera icon. Below the photo area is a blue button labeled 'SAVE'.

Figure 4.2: An example of inputting a picture from the phone's camera into metadata for a particular system in the NRDC.

manages this problem by storing the changes locally and allows users to commit these changes when they reach an appropriate area. Finally and most importantly, the application cannot function as a singular client side application without any support. The QA Application consists of a client and server with the client utilizing a Model-View-Controller(MVC) architectural pattern and the server utilizing a microservice architecture.

The client-side application utilizes a MVC architectural pattern as shown in Figure 4.3. This architectural pattern was executed with Angular.js framework and the C# and Javascript programming languages. The model section of the client-side application is the central nexus of interaction with the main NRDC System. This contains routines that would handle the HTTP communication with the server end, as well as the manipulation of locally stored data. The view section is tasked with the primary task of satisfying the first requirement and showcasing the data in a hierarchal format. In this section, the functionality to navigate through the hierarchy, view imagery, and handle conflicts syncing with the database is handled. Finally, the controller section serves the primary master and control portion of that application that dictates the behavioral actions that result from the interactions made by the user. This section is where the application would issue the command to shift the page views, create the transitional effects in-between views, and initiate the interaction with the model section.

The server side of the application utilizes a microservice architecture where each web service is independent of each other and are combined to create a greater functionality. These services can each be classified into three main groups: Edge Microservices, Administrative Microservices, and Specialized Microservices. The Edge Microservices are infrastructure services that perform the role of data provider from the data base to the client application. The Administrative Microservices perform the role of security and provides varying level of access to the hierarchy, based on the individuals status within the project. The Specialized Microservices provide complex functionality to the application that outside that of the Edge Microservices. These

functionalities range from photo compression and searching to file conversions.

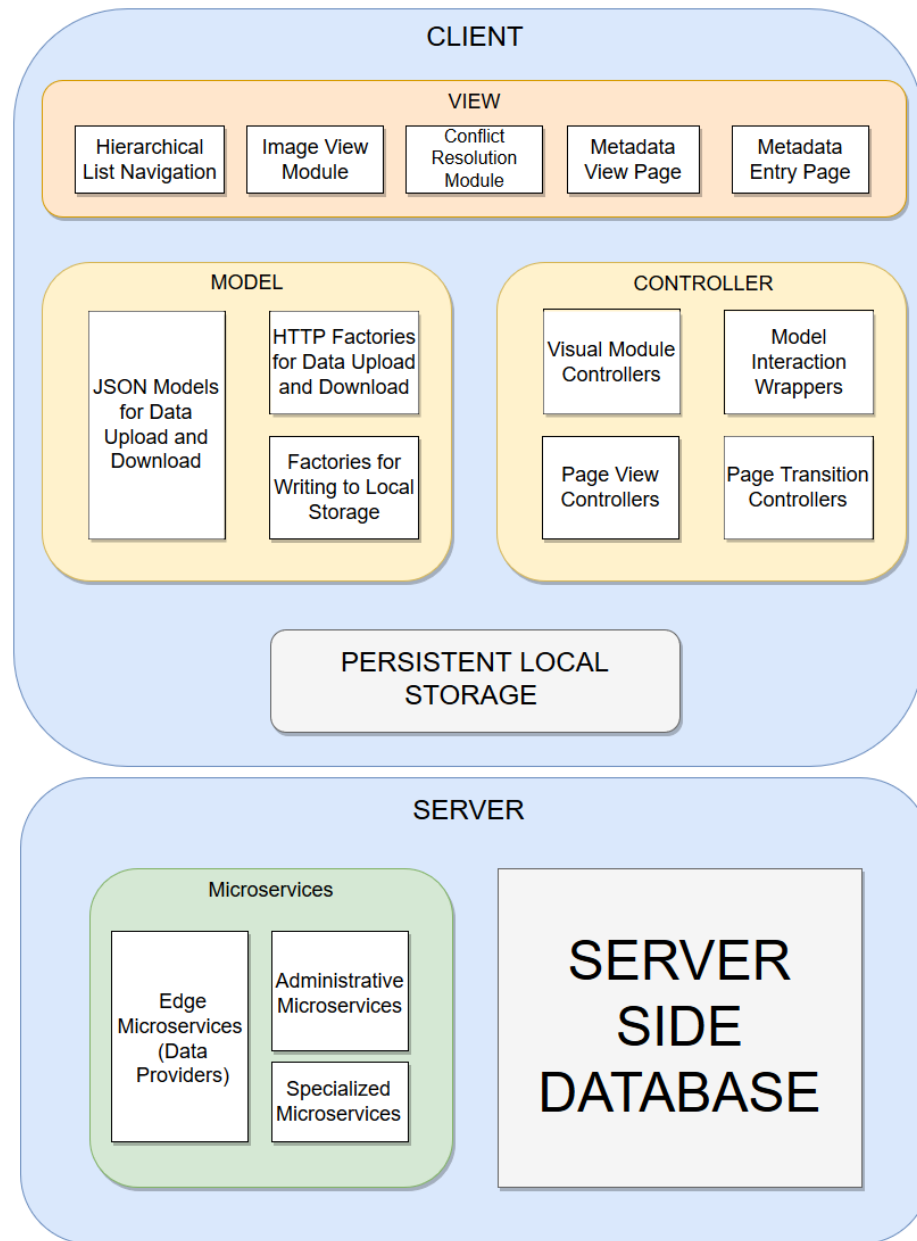


Figure 4.3: High-level block diagram detailing the architecture of the QA app. Client and server are connected via http calls from the mobile application to the Edge Micro services.

UI Design

Throughout development, the User Interface design of the Quality Assurance Application transformed drastically. Initial designs for a prototype implementation of this application, visible in Figure 4.4, were relatively simplistic, utilitarian and focused on the highest level of sensor network metadata that would be managed: a project. A project described a specific cluster of in-situ sensors networked together that collected data associated with answering the question of a single research project. Given the broad scope of a project, the metadata collected and stored was general in scope, requiring only one page and a few form fields. As development on this application expanded so too did the UI design to better accommodate the technical and personal needs of stakeholders.

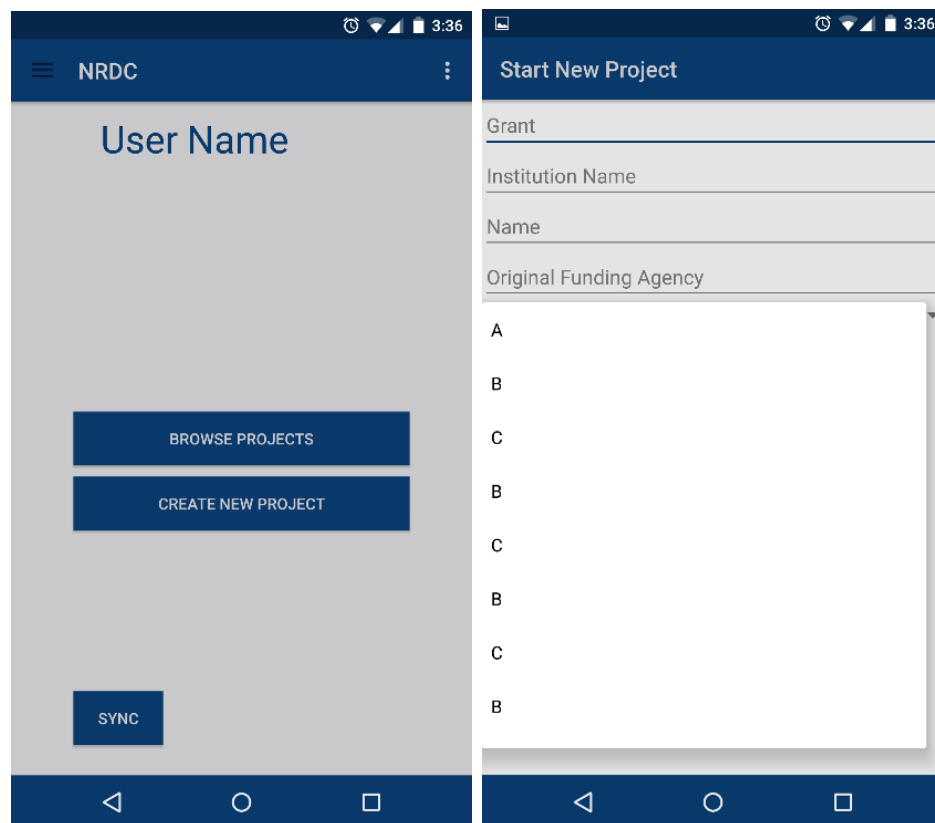


Figure 4.4: A mock up showing the design of the flag dashboard (left) and the data visualization dashboard (right).

The first major design shift was a visual one. The application transitioned out of the simple Flat Design of the initial prototype into a Material Design variant with the addition of more primary color contrasts, subtler form fields and floating action buttons. This design choice gave the application a more modern and commercial feel that people have come to expect from a high quality mobile application.

A second design shift, more critical to the proper functioning of the application itself, was the inclusion of a hierarchical navigation structure. While the details of this structure are explored in Section 4.2.3, from a design standpoint it was crucial to enable the fast traversal of this hierarchy by making each item on every navigation level a large, clickable button that only leads to a sub list of items contained by the item clicked. Figure 4.5 contains an example of one of these sub lists. By placing a link to the information about the previously clicked item in the top right corner, the design of this list navigation negates the possibility of users unintentionally opening information pages that will slow their navigation. This streamlined design enables users to find the lowest level component they want in seconds, and then open information pages for the editing and viewing of related metadata.

Finally, stakeholders expressed a strong need for the display of saved pictures showing the makeup of a sensor system or the structure of a specific component. This need drove the development of a dedicated image viewing component which retrieved images from a remote or local source and displayed them on appropriate pages. The addition of this component required significant planning as each image had many management functions associated with it: saving the image to the database, saving the image locally, delete locally, delete on the database, open an enlarged view of the image, and more. This suite of functionality had to be added without cluttering the limited real-estate of a mobile screen. We overcame this problem with the inclusion of a floating action button which expanded into an array of buttons that each perform one of the above stated functions.

Finally, the interface provides users with a simple, straightforward medium enabling quick access to the forms required by technicians. Visually reminiscent of

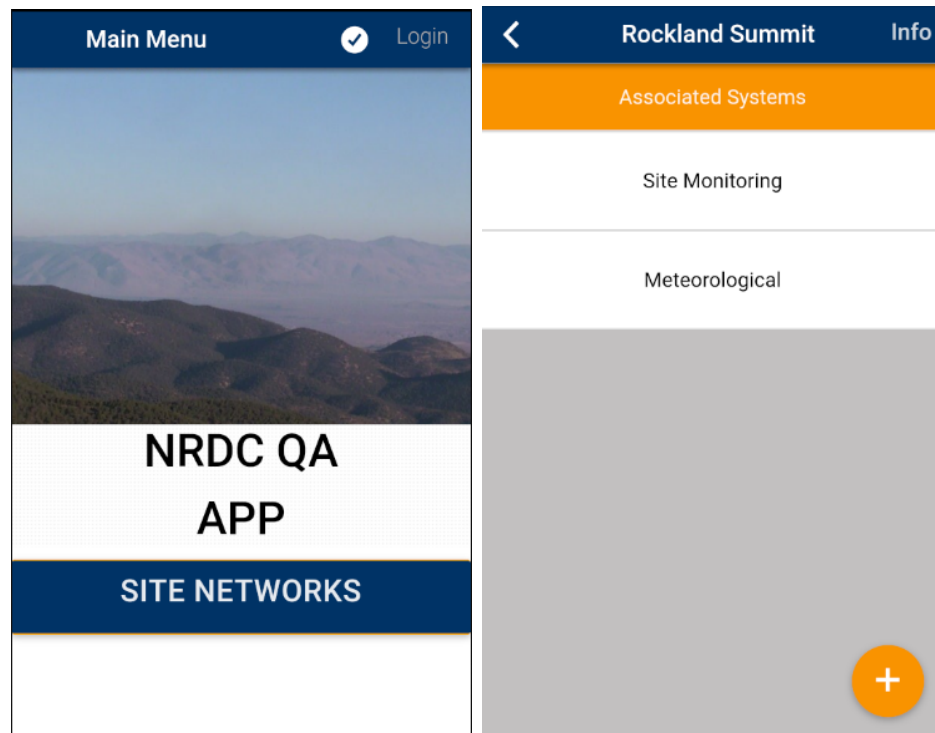


Figure 4.5: Screenshots indicating the final UI design of the Quality Assurance Application. On the left is an updated and ascetically pleasing main menu screen. On the right, we see an example of a sublist in the hierarchical navigation structure. Here we see that Site Monitoring and Meteorological are systems associated with the Rockland Summit Site. Clicking on either item will populate a new list of Deployments that are in the chosen system.

Googles material design, this application takes cues from public facing software to provide a refined interface to encourage smoother adoption of this app among unskilled mobile technology users. Large buttons and clickable lists simplify use for technicians wearing gloves when performing maintenance on sites in high elevations or in colder months.

4.2.3 Implementation Details

Currently, the NRDC QA Application is comprised of a front-end system developed in the Ionic Framework, and a back end comprised of essential and independent web services communicating through a centralized hub [56]. The data transferred between the two are stored inside a Microsoft Virtual Environment with Microsoft SQL Server 2012 as the primary database management. These two main components together allow for a seamless interface between the main databases and the client application.

For the front-end system, the QA application was built with the Ionic Framework [52]. This framework utilizes HTML and CSS as a wrapper to manipulate the interface, as well as Javascript to apply functionality. Once completed, the HTML, CSS, and Javascript are then compiled into the appropriate codebase: either Apple or Android.

A wide collection of libraries and modules were used to simplify development at various stages of implementation. Primarily, Googles AngularJS was used as a structural framework for the Javascript codebase and allowed for a more object-oriented approach to manipulation of the HTML and interaction with microservice APIs [45]. Additionally, Node Packet Manager (NPM) and Bower were used as the main package managers for this application. They ensured libraries, assets, and utilities were regularly updated and organized.

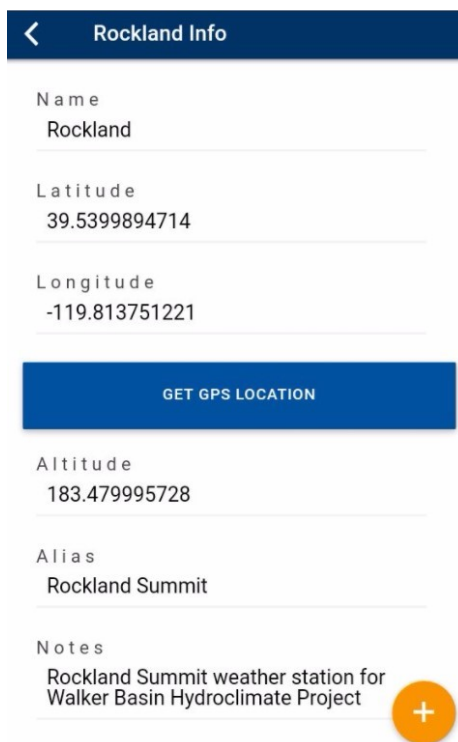


Figure 4.6: An example of retrieving latitude and longitude coordinates from the phones GPS.

Organization of the QA app follows a pattern representative of the hierarchical organization of existing sensor networks managed by the NRDC and associated institutions. At the topmost level, the application presents the user with a selection of Site Networks: a representation of several data collection sites connected by their similarity of purpose or project associations. From there the user selects a site associated with that network, a system associated with that site, and a deployment associated with that system, ending with a hardware component associated with that deployment. This workflow of tunneling down into atomic components gives data scientists a logical means of quickly locating the exact sensor network element they seek by leveraging their knowledge of existing infrastructure.

At any point in the navigation of the sensor network hierarchy, a user can add a new entry to the list of displayed entries or view the details of existing ones. Whether choosing to display or create, the user will be greeted with the same page. If displaying

data about an existing item, the page will be populated with data about the selected element. If the user chooses to create a new item, the form fields on this page are blank and ready for input.

On the element creation screen, visible in Figure 4.6, the user inputs information into blank form fields that expand or contract to fit the size of the input data. The user can also choose to upload a related picture or get their location via their phones GPS. This functionality enables field technicians to upload accurate location data about sites they are working on with the touch of a button. Once all necessary information is entered, the new metadata entry can be saved locally. And, once the user is done adding new entries, they can upload them en masse to the server for storage in the database.

On the view screen, the user is presented with a few different options compared to the creation screen. Principally she can no longer save an entry, only edit or delete with proper permissions, and there appears a floating orange icon in the bottom right

Dynagage Sap Flow Sensor Info	
Name	Dynagage Sap Flow Sensor
Manufacturer	Dynamax
Model	SGA2-WS
Serial Number	135972
Vendor	Dynamax
Supplier	UNR
Wiring Notes	
Installation Details	

Figure 4.7: An example data entry page containing info about a single data sensor. The floating action button in the lower right-hand corner provides the option to add a service entry when clicked.

corner visible in Figure 4.7. From the submenu which this button populates, users can add two different types of metadata about an entry, a document and a service entry. Documents allow users to add related files to a metadata entry. Service Entries are entered when scientific equipment is repaired or replaced.

5 Quality Control

5.1 Role in Keystone

The Near Real Time Quality Control(NRAQC, pronounced “na-rock”) application detailed in this chapter fulfils the intended purpose of the Quality Control module in Keystone through its automated and integrated Quality Control testing process and the creation and binding of QC metadata called “flags,” which indicate specific problems with data points. The quality control testing process developed for this software platform performs several well known and commonly implemented tests on datasets to evaluate the quality of those datasets based on simple metrics. This particular process of automated testing elevates data input into the Keystone Model from L0 or “Raw Data” to L1 or “Data with a Basic Level of Quality Control.”

By automating the testing of datastreams and simplifying the process of configuring tests on incoming data, NRAQC fulfills very well the requirements expected of Keystone’s quality control module. In a way similar to the Quality Assurance software detailed in Chapter 4, NRAQC simplifies the process of creating a quality data product by not only testing data sets autonomously but also by binding the error information directly to tested data points through relational flags. In the NRDC database, each measurement row holds a foreign key which can point to a specific flag dictating its quality level.

In addition to providing this value to a final data product, the NRAQC system works in tandem with the Quality Assurance module and the Data Repair module. From the Quality Assurance module, the QC module and specifically the NRAQC system, receives critical contextual metadata that organizes the data which will be

configured and tested. This contextual metadata provides information about the deployment conditions related to a particular data source and can be used to inform what tests should be performed on that particular source. Furthermore, this metadata can be leveraged to provide more intuitive navigation panes like the hierarchical navigating presented in this chapter. It can also be used to make richer and more meaningful visualizations of tested data.

Finally, the NRAQC application, through a test which checks for holes in data streams, creates spaces in the NRDC database where no measurement exists but a time-stamp should be represented. This introduction of holes provides clear opportunities for these holes to be filled via some variety of Data Repair service. Accordingly, it is in this way that the NRAQC software serves the Data Repair service in our Keystone model.

5.2 Requirements Elicitation and Specification

Software requirements were developed from a formal elicitation process where two stakeholders were interviewed to determine the critical functionalities of the system. The first stakeholder interviewed (S1) is a domain scientist specializing in in-situ environmental sensing and data management. S1 was solicited to express the needs of the intended audience for the NRAQC. His experience with existing QC software greatly aided the development of functional requirements for this software package. The second stakeholder (S2) a data management expert and software engineer employed as lead developer for the NRDC, was elicited because of his expert knowledge on distributed systems and scientific software development. His input helped guide development towards a microservice-based architecture so that the NRAQC system could be better integrated with the existing microservices currently handling the collection of measurements and non-QC metadata in the NRDC. In addition, he suggested that the development of a web-based system that manages significant amounts of feedback from autonomous tests raised concerns of failure when using a monolithic system. The current compatibility with an existing system, and the risk of significant

architectural failure, were the largest factors that drove the development of microservices as NRAQC's primary backend. The final stakeholder interviewed (S3) is an expert in microservice development who has developed microservices for the NRDC.

5.2.1 Requirements Elicitation

Stakeholder 1

1. Can you provide us with a QC Breakdown from a user standpoint?

ANSWER: The QC process takes in data from the user and visualizes the data, where it then allows the user to manually or automatically monitor for suspicious data points or trends. These data points or trends can be determined suspicious automatically or through an experts judgement.

2. Ideally, what does this QC Software look like to you?

ANSWER: The users ideally should be presented with a GUI that is composed of a visualization component, as well as a data selection component that ties in directly with each other. Somewhere in this interface would be the flagging and correction component that would recognize and amend data entries. Finally, if possible, the addition of summary statistics of the region would be a possible selection.

3. Have you used any other QC software before? If yes, Can you tell us some things you liked about it? Some things you didnt like about it?

ANSWER: Yes, I have used multiple QC software tools before, such as GCE Toolbox. Most QC software has a very interesting data streaming visualization component, but they dont have a quality control focus at a standards point of view, just at a streaming point of view. These software tools dont provide us with reviewing, persistent flags, related personnel, and time of the flag.

4. Among the common quality control checks which flagging metrics would be most important to be implemented in a prototype of this software? Why?

ANSWER: Date and Time would be the most significant flagging metric to be implemented in this prototype. I say this because Date and Time is the most common metric to cause confusion in a group of technicians performing QC. There have been notable cases of logger devices being set up to receive daylight savings time while other would not, creating a severe disorganization of data when aggregated.

5. Do you think its important that users have the ability to code in/upload their own custom checks on data points or ranges of data points? Why or why not?

ANSWER: Yes, however this would have to a functionality that would be best reserved for much later in the project. The idea of the GUI is to have a supervised check of the data flagging process. Adding custom scripts to set up new functionality is very unsupervised and would be best discussed at a later time.

6. Would the option to upload flat files of data for QC checking (csv, xml, netcdf, hdf) aid your research? Explain.

ANSWER: No, not especially, at least not at this time. The reason for this is because we would want this application focused on operating on a specific framework, like the microservices being developed for it. However, academically, this would be of high value, but would require an immense time sink.

7. What constitutes easy to install scientific software for you? Would you be able to use command line tools to install software? Do you think other scientists who would use this software would be capable of using command line tools to install this software on a server or cloud?

ANSWER: Easy to install scientific software would entail simple double click .exe files where it would auto-install or run. Adding a command line interface would simply confuse the scientific community that it was meant to serve. Scientists in this field would often lack the technical expertise that would be needed to use command line tools.

8. In terms of a future iteration of this software application, what would the QC Application 2.0 look like to you?

ANSWER: QC Application 2.0 would contain a more advance and fluid visualization component, and this could be anywhere from seamless transitions between enormous figures to seamless loading of data points as data streams in. Additionally, QC 2.0 would also have a more refined user interface than the last iteration. There may be room for a user study to facilitate a more intuitive interface, but this is just speculation. Finally, there would long-term trend analysis available for entire data set.

9. With the idea of the QC application being a gateway to further projects, what is next on the list in terms of development?

ANSWER: What happens next is the real scientific analysis, so this would be the advance visualization tools. Now that the data is correct, this opens up a lot of opportunities to really explore the data and find interesting features or trends. Additionally, this would be a prime opportunity to publish the data to a wider audience, such as the weather service. Finally, this raw data would turn into a data product.

Stakeholder 2

1. How should missing entries be handled with consideration to the NRDC Database with Quality Assurance Software?

ANSWER: There are two approaches to the problem of handling missing entries. The first, is to simply insert a new line containing a null datapoint in the table at the missing timestamp. The other is to load all missing data point entries into a separate table dedicated to missing entries. I prefer the latter method as it negates the possibility of accidental overriding an existing datapoint.

2. What are your thoughts on the most efficient way to develop a series of batch

tests for data points?

ANSWER: I would establish a well defined order of tests such that if a data point fails at any point it will be flagged at the failing test and removed from the flagging pipeline. This means no unnecessary testing will be done after a datapoint has already been flagged. If it passes all of them we should probably have a dedicated OK designation in our flag vocabulary.

3. I have read that Micro services often have their own small scale Databases. Would it be viable, with consideration to the existing architecture of the NRDC, to have a small database associated with Quality Control Microservices?

ANSWER: That depends on what is being stored in this database. Primarily we want to avoid redundant storage, so anything that is already stored in the existing NRDC database should not be represented in the QC database. So, nothing in the measurements table, but perhaps user specific information. We also likely have the structure to store that in our database as well however. There may be little point in having a QC specific Database.

4. Do you think there is value in containerizing this quality control application for distribution?

ANSWER: Yes and no. The components being developed as part of the NRDC bundle of applications are not explicitly designed for use with outside database architectures. The idea of anybody picking up this software and integrating it with their system is great however ideally they will pick up the entire NRDC frameworkDB, apps, architecture and all and drop that onto their servers. Its intended to be a large software package thats proven to work effectively with time series data streams.

Stakeholder 3

1. How would you define microservice especially compared to a traditional service?

ANSWER: Microservices as opposed to a traditional services are a collection of smaller services with a simpler functionality. They are expendable and easy to replace while traditional services tend not to be. Each microservice is able to scale independently from each other, which is the most significant difference and how they bring scalability to growing system. And finally, there is a master and control service discovery that serves as a hub to not only locate services but spin up new instances should the need require.

2. What advantages do you think a microservice architecture can bring to a Quality Assurance application?

ANSWER: Quality Assurance and Control applications have a wide array of functionalities that technicians require or at least desire. The sheer amount of traffic and variety that comes with these functionalities would create a stressed environment on the greater system. By switching to microservices, each of these functionalities can be delegated to an individual service. At the same time, it will allow each functionality to persist independent of each other, should a functionality break down.

3. Which technologies (languages, tools, etc.) aid the development of a group of microservices? Explain

ANSWER: Microservices themselves are relatively new and people are just now beginning to get a handle on that question. However, out of trends from personal experience and those mentioned inside various academic papers, one of the most important technologies is a web framework to actually begin constructing the services. Another important technology is any form of web development language, such as Javascript, C#, or Python. Finally, devTools, container

software, and continuous deployments are growing in number and are considered quite vital in industry.

4. What does it mean to dynamically spawn a microservice?

ANSWER: Microservices are actually design to fail at specific times, believe it or not. When planning for these failures, a common practice is to actually sense for a service failure and create an instance of the same failed microservice autonomously. The alternative would be to manually create a new instance again and that would be grossly inefficient. In certain industries, systems would actually sense that one service is lacking to cater to user demands and would create another instance of a microservice to help manage incoming traffic.

5.2.2 Functional & Non-functional Requirements

The requirements determined by the elicitation process were split between *functional requirements*, which describe the overall technical functionality of the system, and *non-functional requirements*, which outline constraints on the system. The functional requirements of NRAQC were broken down into two priority levels. The base level requirements indicate the critical functions that are vital to the system operability, while the second level requirements represent adjunct functionalities that are not necessary for operability.

From stakeholder elicitation it was determined that the NRAQC system depends on thirteen fundamental operating requirements, visible in Table 5.1. In addition to these thirteen baseline requirements, NRAQC documentation also specified eleven secondary requirements and eight tertiary requirements, specified in Tables 5.2 and 5.3, respectively.

In addition to these functional requirements, the NRAQC system was developed under the constraint of eight non-functional requirements. The first requirement dictates that the NRAQC system is built as a web application. The second requirement describes the backend microservices being written in Python with the Flask

Table 5.1: The core functional requirements of the NRAQC Application.

Requirement	Description
FR1.1	The application will utilize a set of backend microservices to communicate to a greater database.
FR1.2	This application will monitor a database table containing time series measurements for when new values are stored.
FR1.3	This application will ingest data points from an external database autonomously when new values are detected.
FR1.4	This application will run each data point through a series of tests specific to its measurement type.
FR1.5	This application will modify a data member associated with the tested data point indicating its checked status called L1 flag.
FR1.6	This application will, when a test fails, will set the L1 flag as a failure with additional information indicating which test it failed.
FR1.7	This application will provide to the user an web interface which lists flagged datapoints.
FR1.8	This application will provide to the user an web interface which visualizes a data stream as a line graph.
FR1.9	This application will clearly indicate flagged data points on the data visualization with a coloring distinct from unflagged datapoints.
FR1.10	The application will handle the login and authentication of administrative staff and technicians.
FR1.11	The application will capable of storing user-specified changes to data and streaming management.
FR1.12	The application will provide administrators the appropriate editing rights to associated data streams while limiting said rights of lower-level users.
FR1.13	The application will provide a web interface for the administrative functionality of this application.

library as its main framework [82]. The third requirement indicates that user interface must be written with HTML, CSS, and JavaScript, with Angular4 as the primary framework [45]. The fourth requirement ensures that the data model followed by the NRAQC system coincides with the model followed by the greater NRDC system. The fifth requirement describes the use of containerization provided by Docker. The sixth requirements relates to the fifth in that the system was developed inside the Linux operating system. The seventh requirement necessitates that the flagging subsystem can detect the availability of multi-core processors within its backend environment.

Table 5.2: The secondary functional requirements of the NRAQC Application.

Requirement	Description
FR2.1	The application shall provide users with the option to force a check on selected data points.
FR2.2	This application shall bundle associated flags into a flag set to minimize redundant listings on the flag display interface.
FR2.3	This application shall provide users with a robust interface for data source configuration.
FR2.4	This application shall provide to the user detailed information about individual data points when hovering over the data visualization.
FR2.5	This application shall create a report of flagged data points over a certain time period.
FR2.6	This application shall distribute a flag report to a data steward via email or compatible messenger program.
FR2.7	This application shall provide users with an interface to discover similar data streams and overlay them over the currently visualized stream for comparison.
FR2.8	This application shall enable users to save data streams for quick viewing on a dashboard.
FR2.9	This application shall enable a user to validate flagged data by manually setting a L2 flag.
FR2.10	This application shall provide functionality that allows a user to save L2 flags and write them to the database.
FR2.11	The system shall keep a running audit of which user set validated which flagged datapoints.

Table 5.3: The tertiary functional requirements of the NRAQC Application.

Requirement	Description
FR3.1	This application may provide functionality for the upload of a flat file containing time series data.
FR3.2	This application may run quality control tests on an uploaded flat file.
FR3.3	The application may write out a Quality Controlled flat file containing the original data points with appended flags.
FR3.4	The application may provide the user with a means to download the Quality Controlled file.
FR3.5	The application may provide functionality for the upload of custom checks in the form of coded functions.
FR3.6	The application may provide security measures to prevent the injection of malicious code.
FR3.7	The application may provide functionality for expert assisted QC with functions for long term trend analysis and scientific modeling.
FR3.8	The application may utilize machine learning for intelligent flagging to reduce false positives.

5.3 Software Design

Following a systematic approach to software development, a formal design processes was undertaken after gathering requirements from stakeholders. The following subsections detail a subset of the designs procured for this software and can be used as reference when attempting to modify or reproduce the NRAQC software.

5.3.1 High Level Design

For the high level design of the NRDC QC software we have developed two documents representing two parts of a client-server architecture: a website map and a component diagram. Pictured in Figure 5.1, the website map of the QC Application shows the intended layout of the Quality Control Application's main ui. After being prompted to login from a landing screen, users will begin at the flagging dashboard. From here users can navigate to either the Data Visualization page, the Administration page or

NRDC QC Webmap

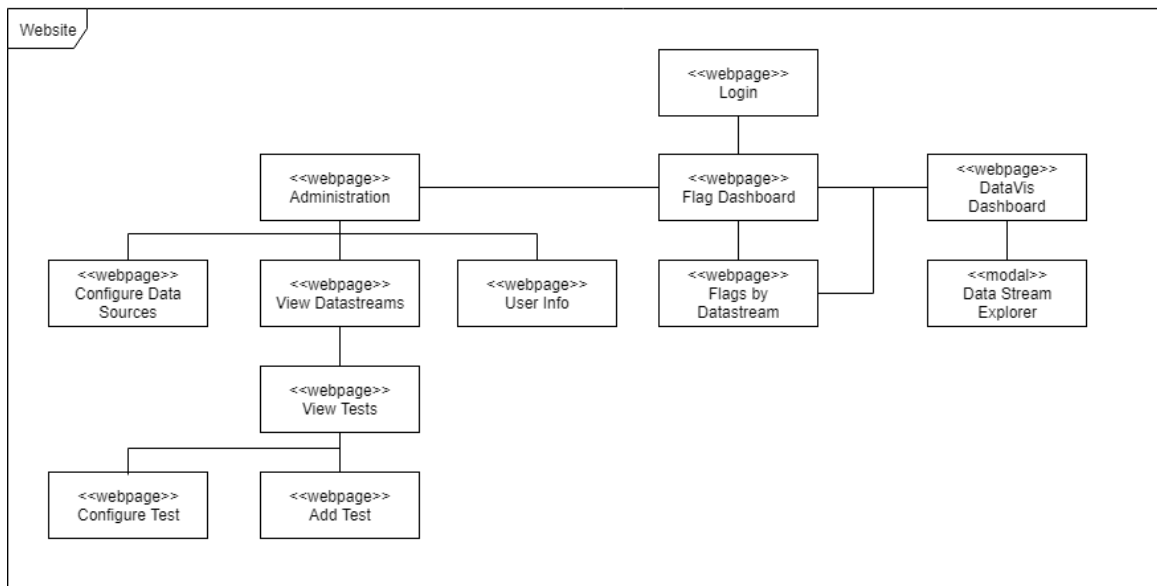


Figure 5.1: Website map for the NRDC Quality Control Application.

click on the flag visualization for a more detailed breakdown of flagged measurements by data stream. From each respective top level page users will have links to sub-pages or modals that provide information about the application, provide tools to configure the application or provide some sort of relevant service. More detailed information and visual mock-ups on these pages can be found in the UI Design portion of this document.

The second design document provided is pictured in Figure 5.2 and shows a component diagram describing the interaction of server-side microservices supporting the UI and providing automated measurement testing functionality. As every component is implemented as a microservice it should be noted most interfaces between components will be implemented as RESTful APIs. Each service in the edge service frame

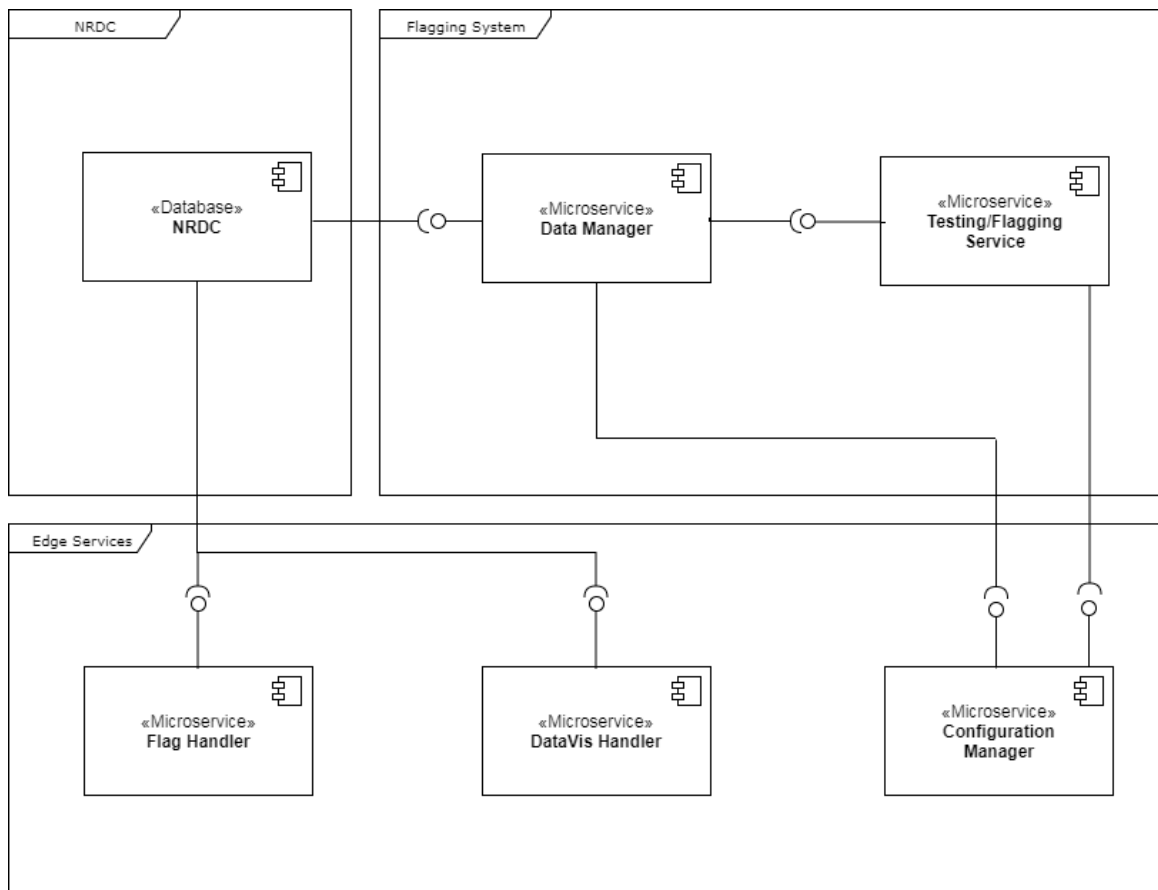


Figure 5.2: Component diagram showing high level architecture for server side of NRDC Quality Control Application.

is assumed to be connected to the internet and provide RESTful API's exposing the functionality of the QC app to client side code. To give a comprehensive understanding of each component and it's role they will be enumerated and explained in detail for quick reference.

NRDC (Database)

This component will not be implemented as part of the Quality Control Application, however portions of it may be modified to accommodate specific requirements of QC metadata. It principally represents the necessary data source that is required for this QC system to operate. The system is being developed to accommodate an exchange of this component with an remote API or a flat file system. At the most fundamental level however the Quality Control system is designed to work with direct access to a database with a known table structure that conforms to certain open data standards.

Data Manager (Microservice)

This component will be implemented as a micro service and essentially provides all functionality of retrieving, formatting, disseminating and writing measurement data, flag data and related metadata. This component provides a flexible *interface* to the data source that can be reconfigured to accommodate an API data source or a flat file system. It requires the implementation of two other *interfaces*: one which allows it to send and receive data from the **Testing/Flagging Service** component and another which allows it to receive data from the **Configuration Manager** component.

Testing/Flagging Service (Microservice)

This component provides the core functionality to the automated Quality Control of streaming data. It should run autonomously and have no direct interaction from the user at any time. It provides one *interface* allowing the **Data Manager** to send measurements to it for testing and retrieve flagged measurements from it for writing to the data source. It requires the implementation of one *interface* from the **Configuration Manager** to allow the upload and modification of tests from the

client.

Flag Handler (Microservice)

The flag handler, as an edge service, provides an interface between the client and server. It's primary role is to handle JavaScript code which queries for flags. It will filter, format and bundle flags so that they may be quickly rendered upon being downloaded by the client. Formally, it implements an *interface* between to connect it to the **NRDC**.

DataVis Handler (Microservice)

The DataVis Handler, as an edge service, implements an *interface* to connect with the **NRDC** component and retrieve the desired information. This microservice handles all requests from the data visualization component of the UI and performs necessary formatting and organization to ensure that the Web page can focus on rendering data points instead of wasting power processing them. A key functionality of this service is the ability to classify flagged data points as distinct from normal data points so they will be rendered differently by the data visualization component.

Configuration Manager (Microservice)

This component provides functionality to the client of receiving user configuration requests and sending down stored user configuration info. In addition to these client-server interactions, the **Configuration Manager** implements *interfaces* to distribute configuration objects to the **Data Manager** component and the **Testing/Flagging Service**. It can also notify the two services of a change to the configuration and request a temporary cease of the automated flagging cycle while re-configuration occurs.

5.3.2 Medium Level Design

For the medium level design, the NRDC QC software can be represented in the form of a comprehensive class diagram. Pictured in Figure 5.3, the class diagram depicts the

overlying class hierarchy of the NRDC QC system. However, for added convenience, this diagram can be split into four sections that roughly equate to planned components: the Configuration, the Datasource, the Test Suite, and the Edge Classes. The Configuration will contain the classes that start and maintain changes to the system. The Datasource will contain the classes that entail the reading and writing of data. The Test Suite will contain the classes that make up the testing and flagging portion. And finally, the Edge Classes will contain the classes that interact with the client side of this application. A detailed description for each is included below.

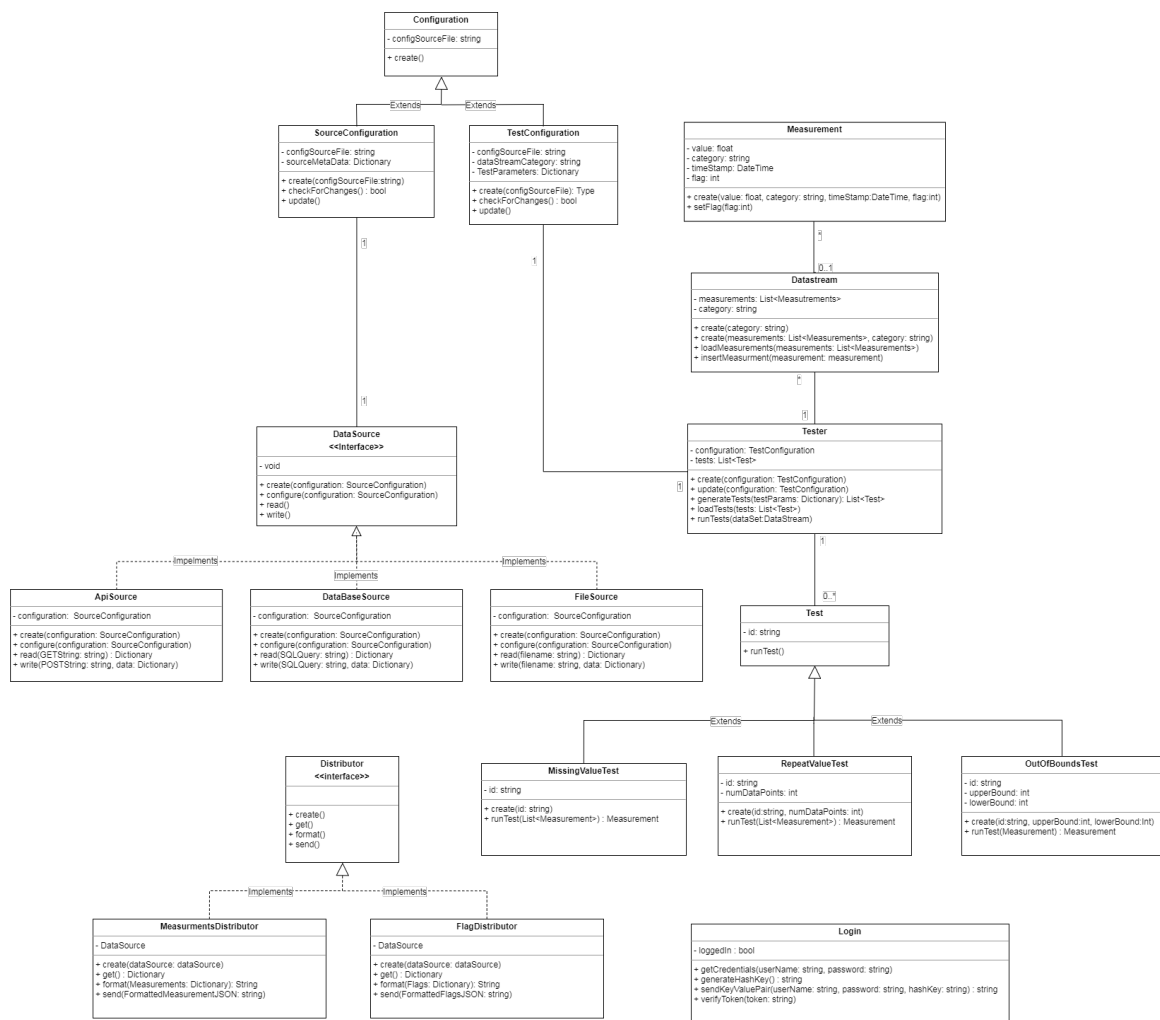


Figure 5.3: Comprehensive Class Diagram for Quality Control Application.

Configuration

The configuration class is primarily in charge of maintaining the various configurations required to run the NRAQC software platform. In this diagram (Figure 5.4) two classes are defined for the two key components of NRAQC which can be configured: testing and data sources. They both inherit from a generic configuration interface class.

The testing class manages configurations which define tests performed upon measurements. Source configuration, by comparison, is in charge of managing the configurations associated with data sources for the NRAQC system. Through the source configuration class, users should be able to define where their data comes from and how to access it.

In addition to passing around configurations to other modules, these classes are called by the Configuration Manager microservice to help with the writing, reading and parsing of configuration files when requests are sent to the microservice. These configuration files are written in XML. Examples of them can be seen in Listing 5.1 and 5.2.

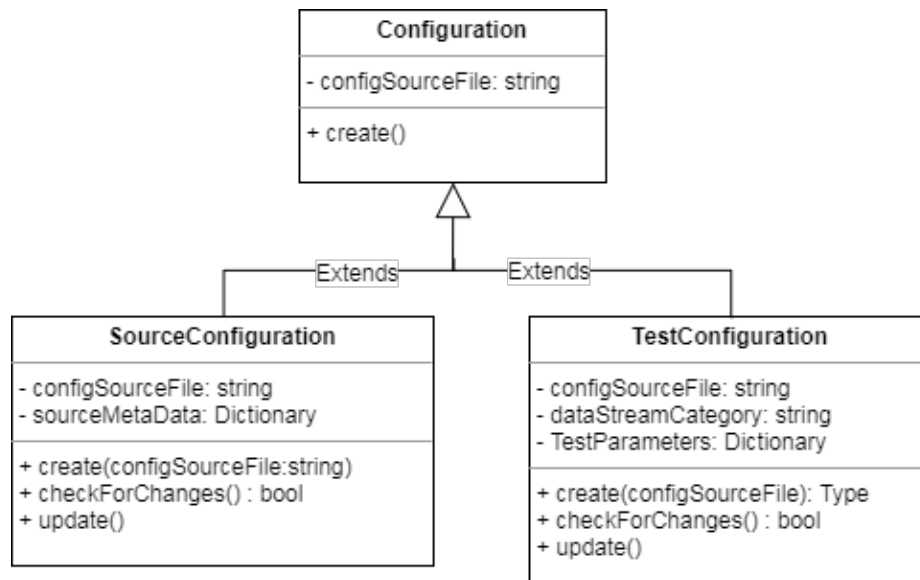


Figure 5.4: The Configuration section of the Class Diagram.

Listing 5.1: Example of test configuration file.

```
<DataStreams>
  <DataStream>
    <Stream>63</Stream>
    <Test>
      <Type>Bounds</Type>
      <Max>28.0</Max>
      <Min>-40.0</Min>
    </Test>
    <Test>
      <Type>Repeat Value</Type>
      <RepeatThreshold>3</RepeatThreshold>
    </Test>
  </DataStream>
  <DataStream>
    <Stream>66</Stream>
    <Test>
      <Type>Bounds</Type>
      <Max>28.5</Max>
      <Min>-19.7</Min>
    </Test>
    <Test>
      <Type>Repeat Value</Type>
      <RepeatThreshold>3</RepeatThreshold>
    </Test>
  </DataStream>
</DataStreams>
```

Listing 5.2: Example of data source configuration file.

```

<DataSource>
  <Type>Database</Type>
  <Language>SQL</Language>
  <Implementation>Microsoft SQL</Implementation>
  <Connection>mssql+pyodbc://. . .</Connection>
  <Username>cilstudent</Username>
  <Password>. . .</Password>
  <InitialCatalog>
    <Name>ProtoNRDC</Name>
    <Tables>
      <Context namespace="Infrastructure">
        <Table id="Deployments">
          <Name>Deployments</Name>
        </Table>
        <Table id="Components">
          <Name>Components</Name>
        </Table>
      </Context>
      <Context namespace="Data">
        <Table id="Data Streams">
          <Name>Data Streams</Name>
          <TestedDataStreams>
            <Stream>63</Stream>
            <Stream>66</Stream>
          </TestedDataStreams>
        </Table>
        <Table id="Measurements">
          <Name>Measurements</Name>
        </Table>
      </Context>
      <Context namespace="Vocabulary">
        <Table id="L1 Quality Control">
          <Name>[L1 Quality Control]</Name>
        </Table>
      </Context>
    </Tables>
  </InitialCatalog>
</DataSource>

```


Datasource

The “datasource” group of classes (shown in Figure 5.5) all implement a single interface called “DataSource.” By having this singular defined interface, each unique data source manager object communicates with the source configuration which is required to define how they should manage their data. While the implementations of these interfaces will be very different, they all behave approximately the same way.

The three different classes defined here manage the three most common sources of environmental data which would be quality controlled for. The API source allows users to define a series of API endpoints which can be used for reading and writing data to and from this software solution from a remote server. The Database

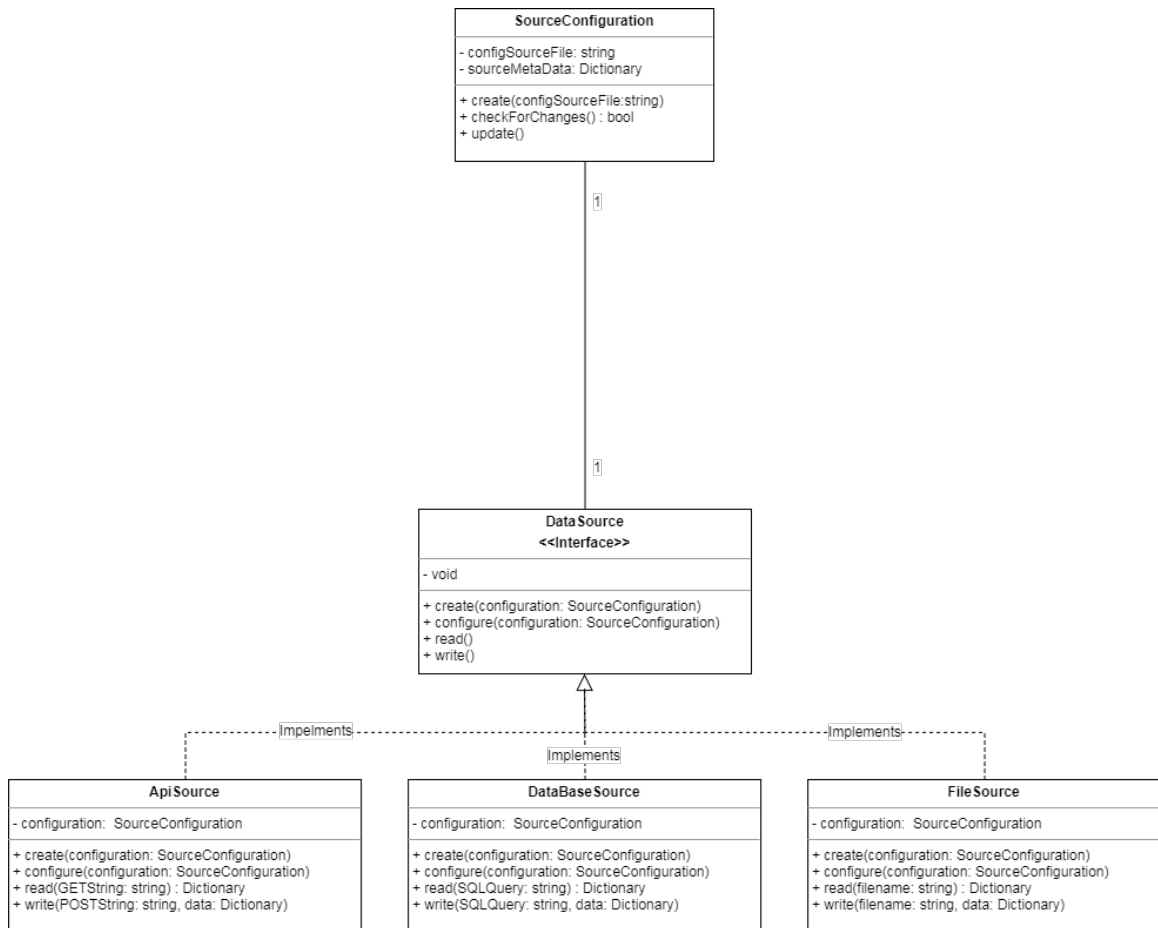


Figure 5.5: The Datasource section of the Class Diagram.

source allows provides tools for the reading and writing of data to a local or remote database with a direct access address and login. Finally, the file source provides tools for the reading and writing of flat files containing time-series data. For this thesis, the DataBaseSource was the only class implemented and it was configured to communicate with the NRDC database directly.

Test Suite

The Test Suite shows all classes necessary for the execution of Quality Control Tests configured by data managers. As can be seen from the diagram in Figure 5.6, the testing classes are driven primarily by a core class called the Tester. The tester manages the configurations passed into it by the TestConfiguration class, generates the needed tests from a number of base classes, ingests and formats data for systematic testing and runs all tests in a predefined order.

In order to ensure our measurements are tested in an expected format, all data is loaded into the Tester class as list of DataStream objects. A datastream object is defined as a list of measurements along with some additional metadata. Finally a measurement is a base object which contains a timestamp, a single float value and a flag. When created, measurements have a *null* value for flag. Upon being tested they will be set with a value corresponding to a particular integer code defined in our database tables.

Extending from a simple interface so that the Tester class can agnostically call tests regardless of their subtype, each test class manages a specific type of test on datastreams.

The **MissingValueTest** checks the timestamps of two data points for a time-difference which exceeds that of the standard difference defined by the time series. For example, if the time series is at 10:00 min intervals, and two timestamps are at 1:20pm and 1:40pm then a missing value has been found. Upon finding this value, this test will generate a new Measurement object (or multiple where appropriate), with a null value at the missing timestamp(s) and store it in a separate list for later

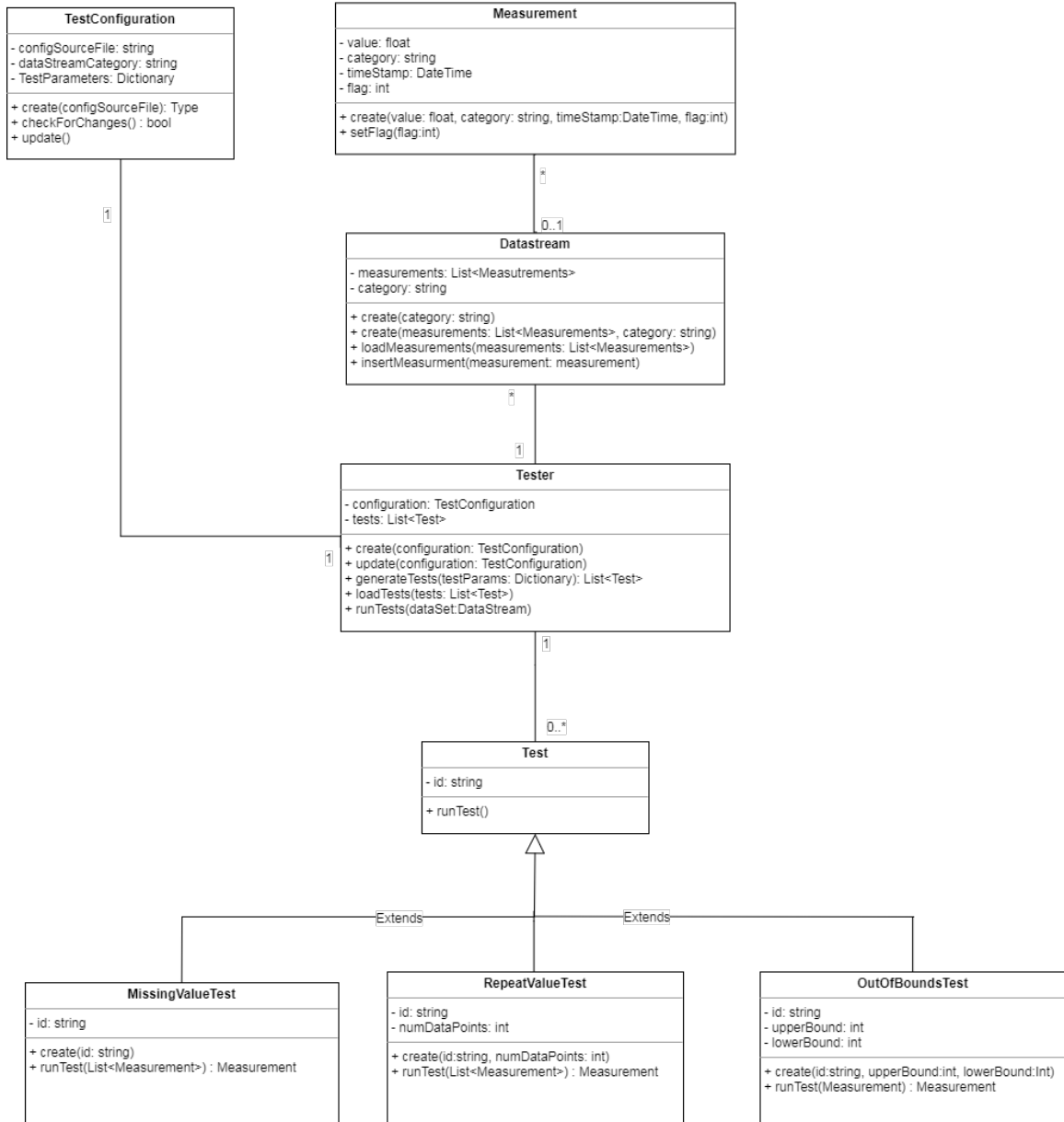


Figure 5.6: The Test Suite section of the Class Diagram.

insertion into the database. This test runs automatically on all datastreams which have other tests run on them.

The **RepeatValueTest** checks a user-defined number of data points which particular a particular data point in time to see if a particular value is a repeat value. So, if a user defines a threshold of *five* repeats for a repeat value, then this test will check the five preceding data values. If any one is different then this measurement goes un-flagged, otherwise it is flagged. This threshold is stored in the test configuration file and loaded in from the configuration class.

Finally the **OutOfBoundsTest** class checks one measurement against user defined ranges of Maximum and Minimum allowable values. Although defined here as integers, they were ultimately implemented as floats. These ranges are stored in the configuration file and loaded in from the configuration class.

Edge Classes

The final collection of classes, shown in Figure 5.7, are a handful of utility classes which manage various components of user interface communication. Specifically the two distributor classes manage the output of measurements for client-side visualizations and flags for various views that require this data. They also perform tasks of aggregation and interfacing with data sources, like the NRDC database. Finally, the login class communicates with software specific database that holds login data and passes back tokens verifying login requests.

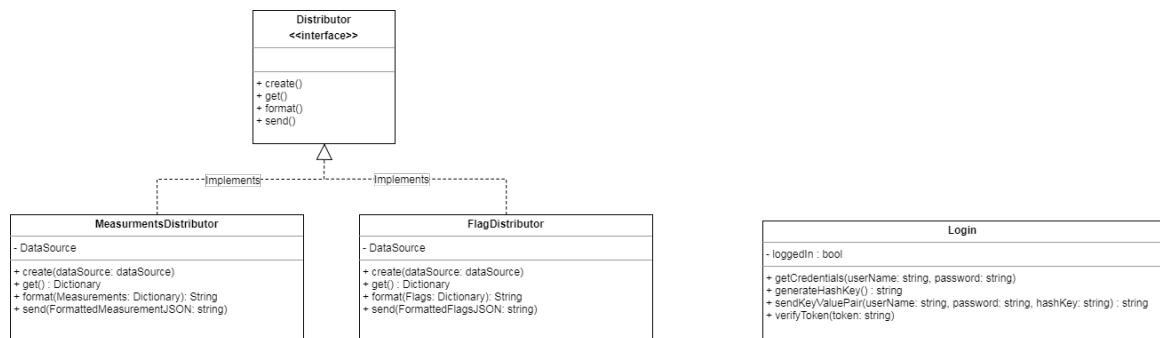


Figure 5.7: The Edge Classes section of the Class Diagram.

5.4 Detailed Design

For the detailed design of the QC Application, the system can be represented in the activity chart presented in Figure 5.8. This activity chart provides a more generalized look at the main QC pipeline. At the highest level, the pipeline shows a fluid process leading with initialization and configuration into a paralleled section depicting the testing and flagging process. However, each piece of the activity chart can be further broken down further into their own subsystem and categorized into two main sections: the Server side and the Client side. Each of these sections will contain an additional number of activity charts according to the number of pieces within the original chart. These additional charts describe the overall flow of the various sub-systems within the greater QC system.

5.4.1 Server Side

The first activity chart, shown in Figure 5.9, depicts the flow of actions leading to the initial configuration of the QC system. The system will read in a master configuration file which will be in the form of an XML document. This XML will be parsed and proceed to be appropriately loaded as a test configuration or a data source configuration. It is then where the objects will instantiate a data source and all tests. These objects make up the crucial settings that will determine the parameters of both corresponding data source and tests.

The second activity chart, shown in Figure 5.10, details the process of retrieving measurements from the NRDC system. This section can be split up into three subsections based off the type of data source. If the data source is an API, the workflow would consist of querying for desired data streams and converting them into the system's native datastream class. It is at that point where each datastream is iterated over and each new measurement can be gathered and placed within. If the data source is instead a formal database, the process would entail querying for datastreams and iterating over each one for new measurements. These measurements will be come

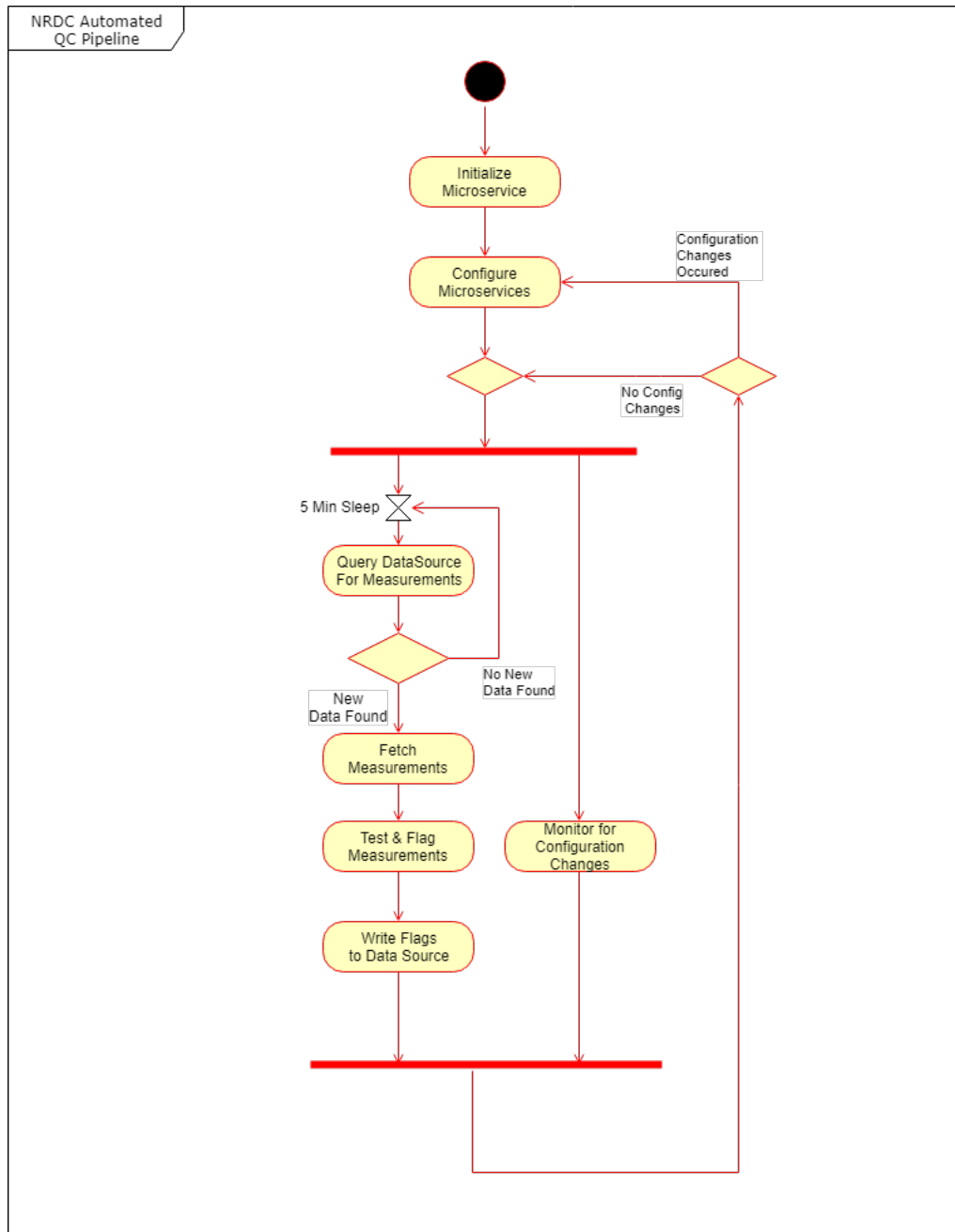


Figure 5.8: General Quality Control Activity Chart.

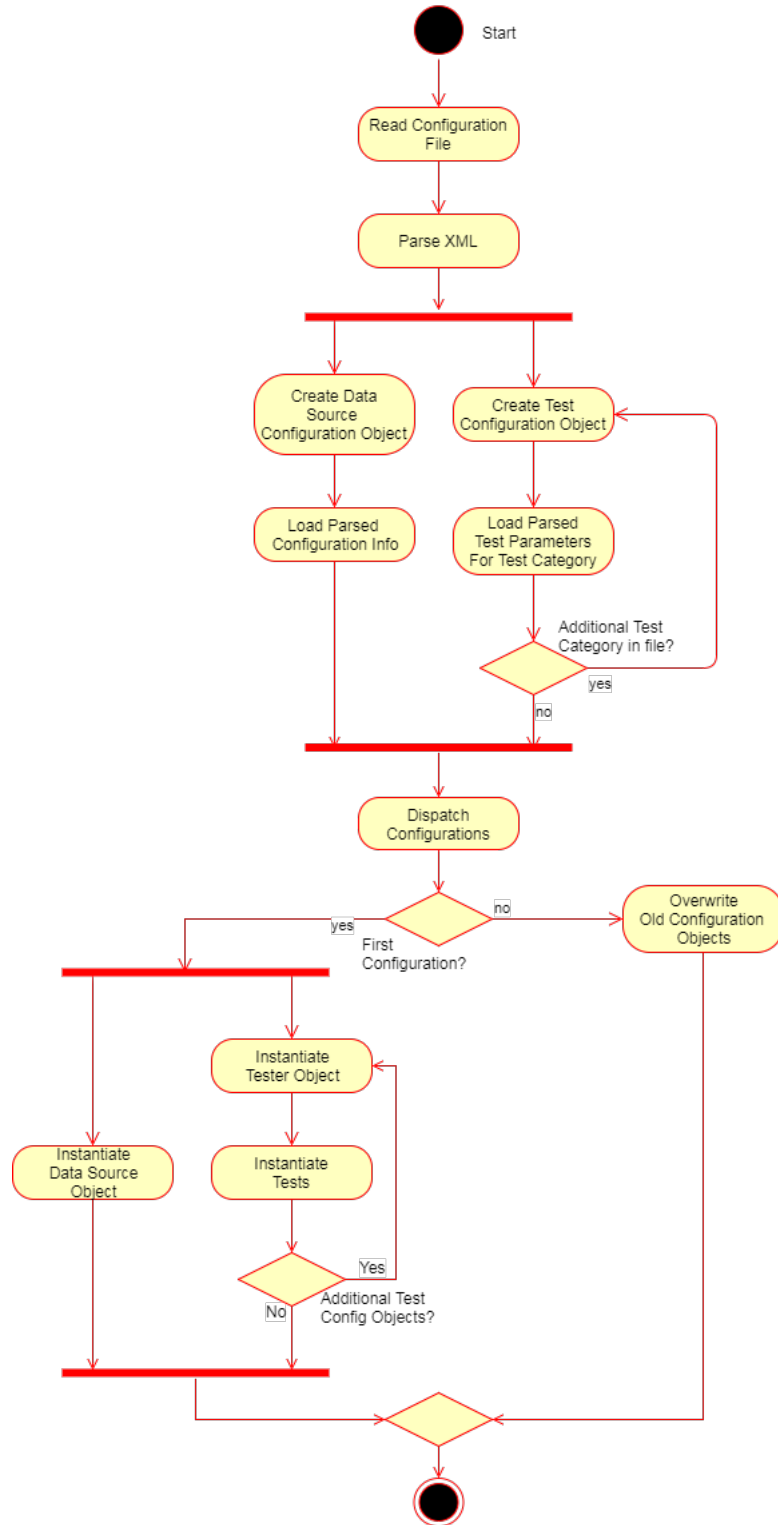


Figure 5.9: Activity Chart for Configuring the Microservices.

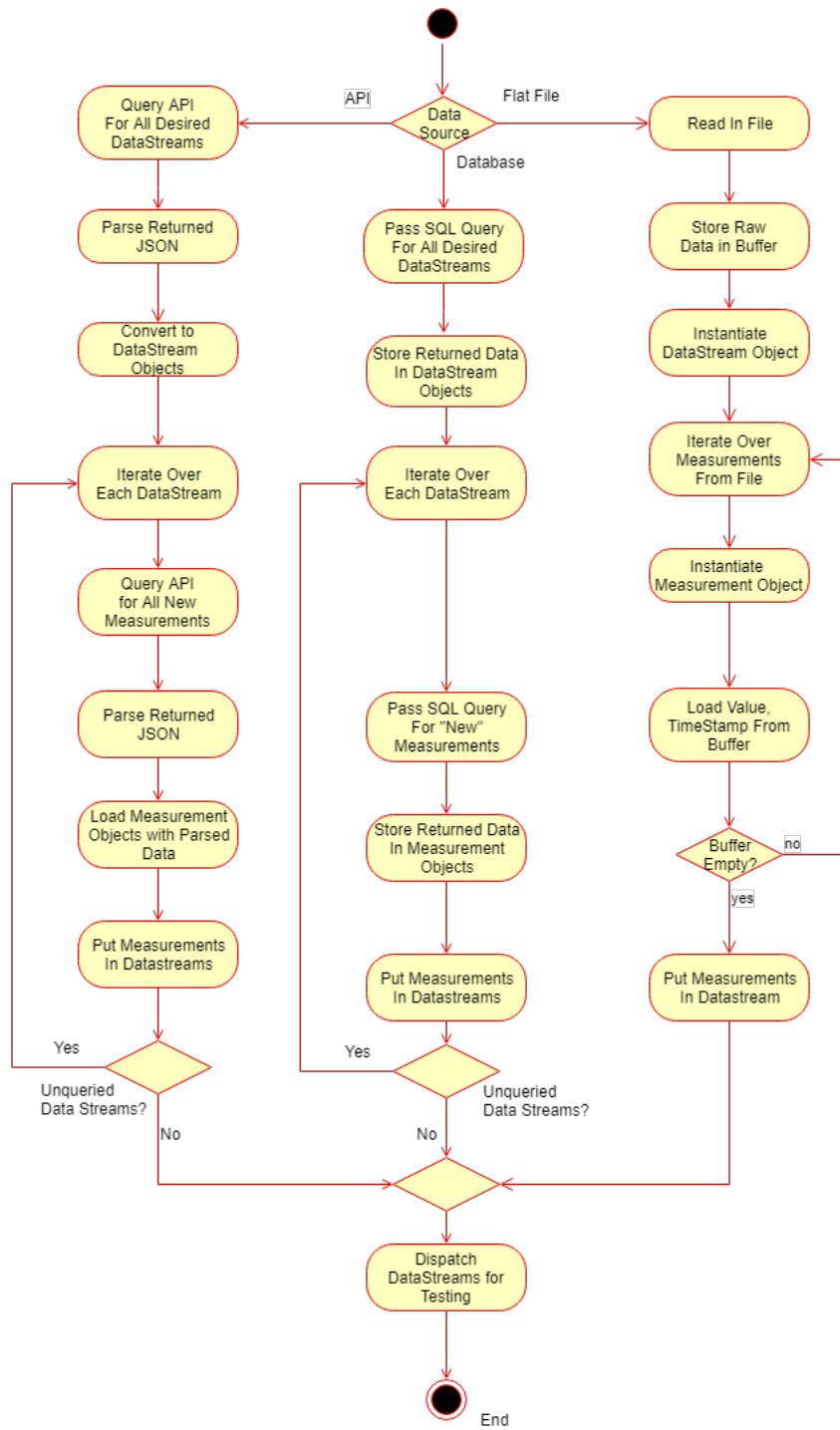


Figure 5.10: Activity Chart for retrieving the Measurements from the NRDC system.

in a raw format and processed into the established Measurement object and placed into the datastreams. Finally, if the datasource is a file, this process would instead simply reading the measurements from the file and instantiating them as measurement objects in the system. These objects would be formatted for timestamps and eventually placed in datastreams. It is at this point in all three sub-section where they converge and send the datastreams off to testing.

The third activity chart, shown in Figure 5.11, describes an outline of an independent configuration check that will be running constantly as the system persists. This check will run every minute and check the configuration file for new changes. The process of checking will be composed of retrieving a cached configuration and the one present within the file. These configurations will be hashed and compared where any changes will be noted and the cached configuration will be replaced with the new one.

The fourth activity chart, shown in Figure 5.12, presents the general method of testing within a testing service. Prior to entering a testing service, datastreams will be organized based on category. After entering the appropriate testing service, each datastream will fetch a measurement. Depending on the measurement, the test may require multiple data points. If so, an appropriate amount of additional parameters will be retrieved. These are then tested for failure, and flagged as either a pass or a failure. The failures will receive feedback based on the type of failure. regardless of pass or failure, these measurements are loaded up into a Post-Test data structure and then sent off to update the database.

5.4.2 Client Side

The fifth activity chart, shown in Figure 5.13, showcases the client side functionality of dynamically generating tests. This method begins when a user navigates to a specific deployments and begins the selection process that would eventually lead to the types of test to be generated. Currently, this systems supports only three types of tests: Bounds Checking, Repeated Values, and Missing Entry. It is here where the

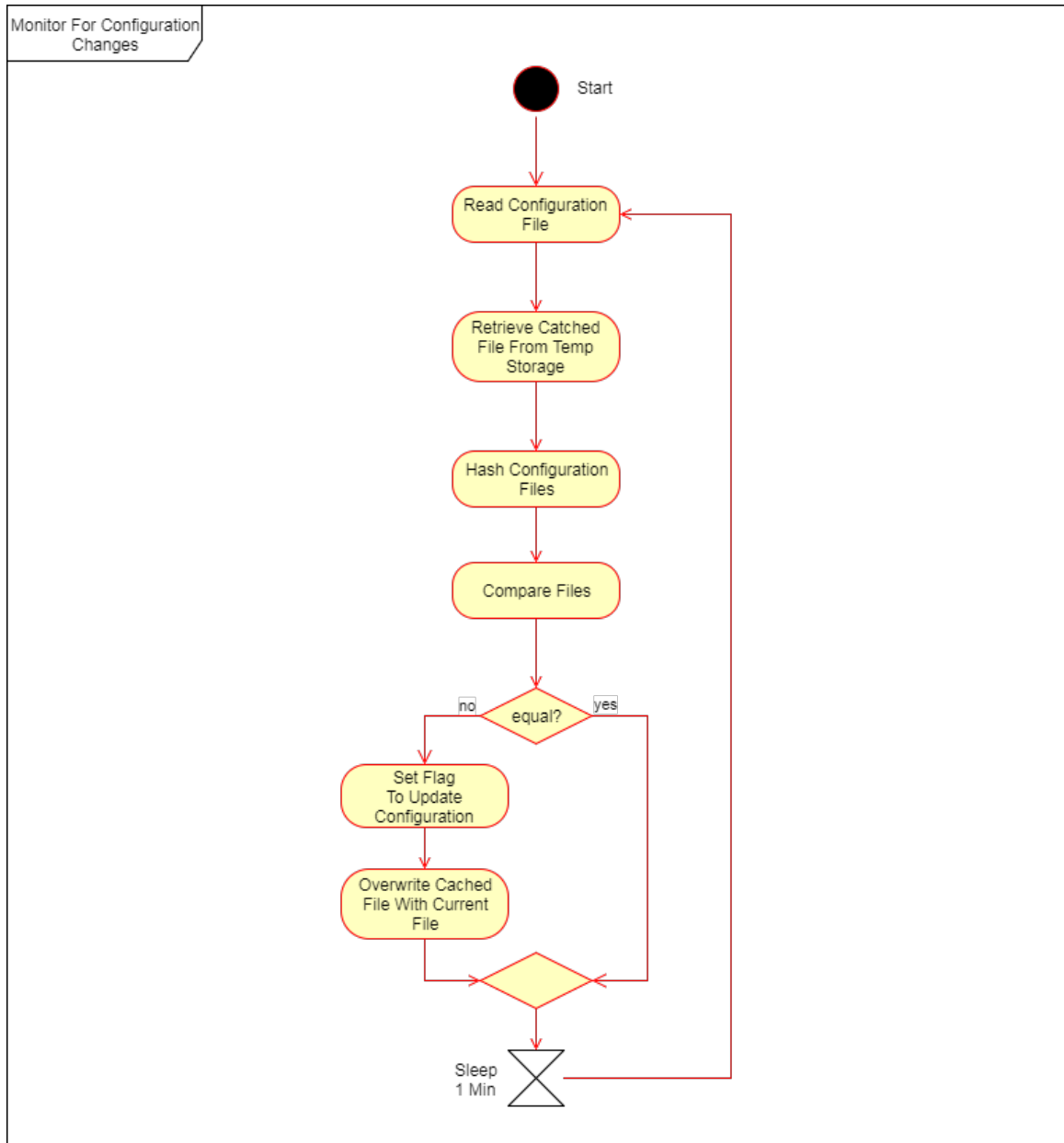


Figure 5.11: Activity Chart for Monitoring any changes to the Configurations.

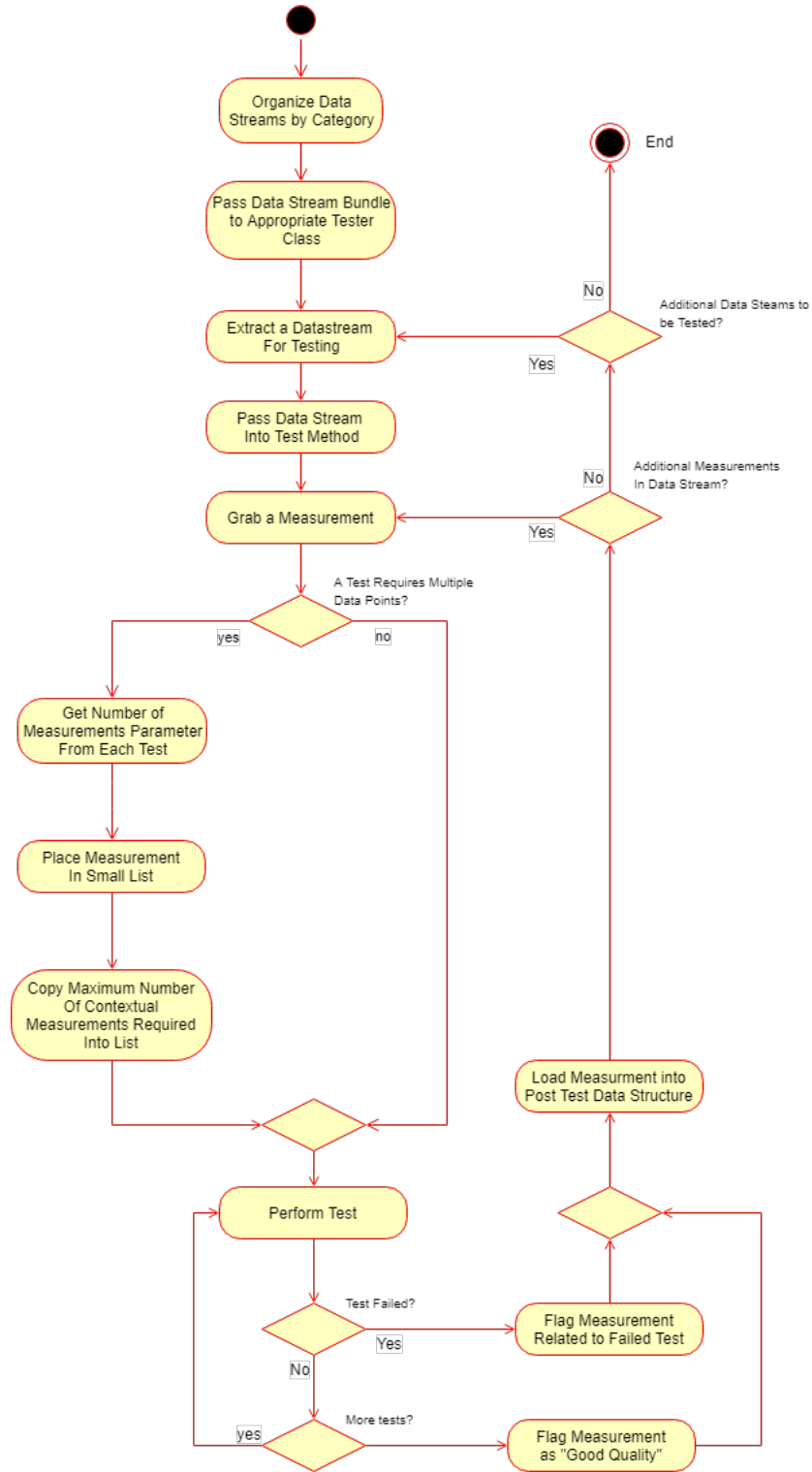


Figure 5.12: Activity Chart for Testing and Flagging of incoming Measurement data.

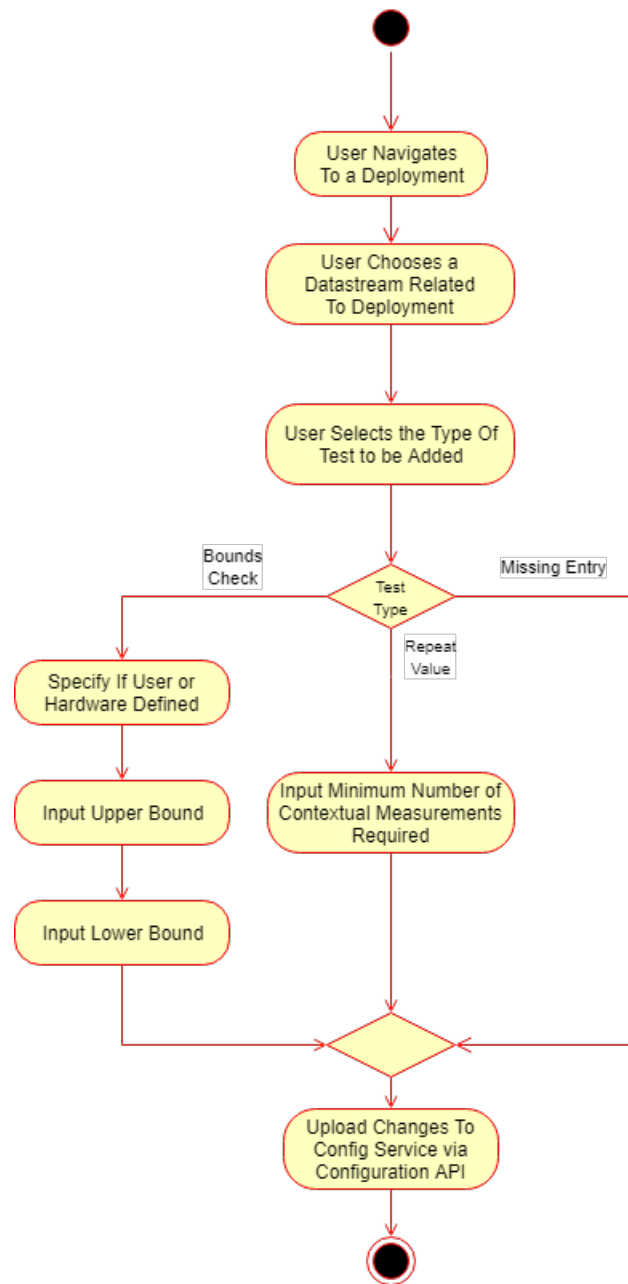


Figure 5.13: Activity Chart for Creating the Tests.

additional checks can be set according to each different type. Once completed, the system will send the configurations off and a test will be generated.

The sixth and final activity chart in Figure 5.14, details the client side functionality of visualizing flagged data. In this activity chart, the user begins by logging in and organizing the flagged data according to the type of flags. The user then can download the measurements and this will render a doughnut chart depicting the frequency of flags. By further examining the sectionals of the chart, this will bring up a bundle of flags. Each flag bundle contains the option of rendering a data visualization in the form of a parallel line graph with unique coloring.

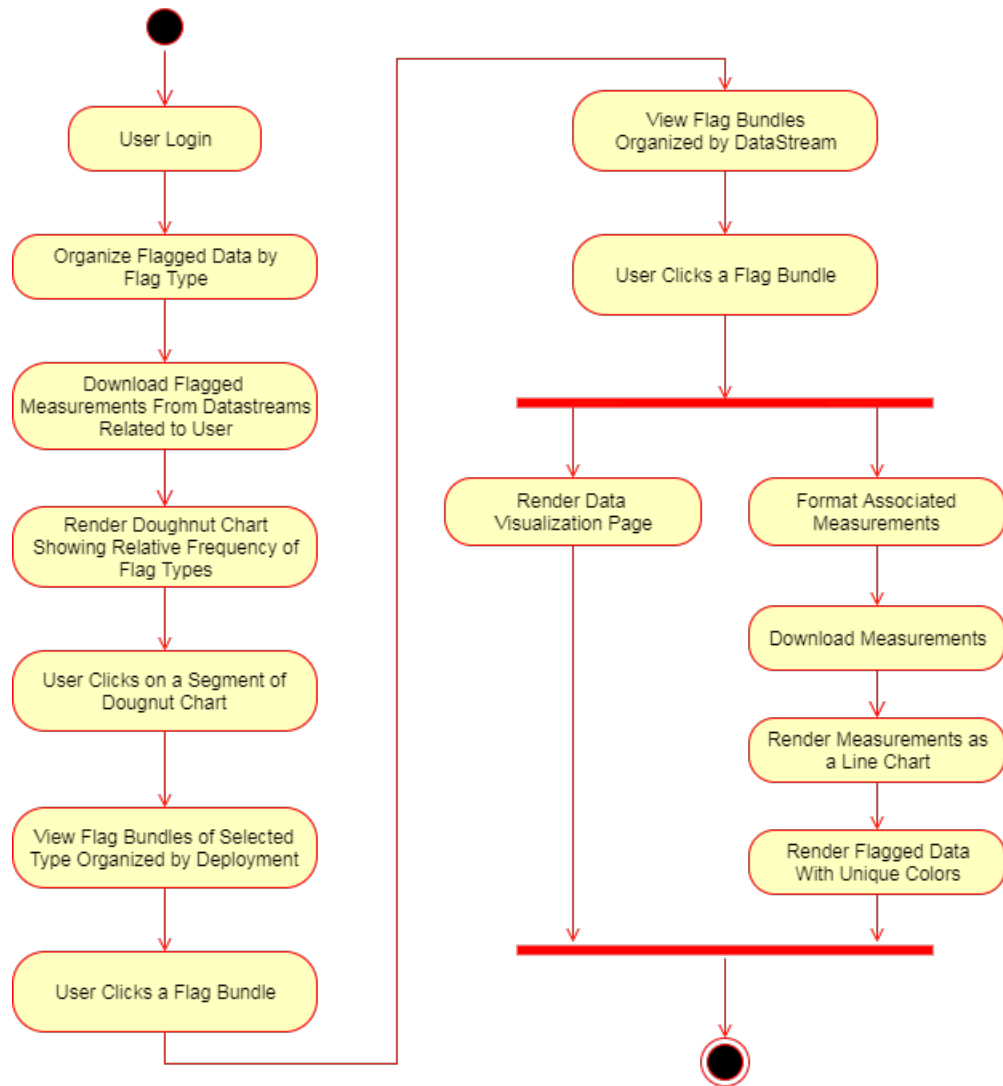


Figure 5.14: Activity Chart for Visualizing the Flagged Data.

5.5 User Interface Design

The User interface for the NRDC Quality Control application was designed as a website containing multiple pages and components that communicate with the remote micro services detailed at length in other sections. In early planning stages NRAQC was designed to have three top level pages which users could navigate between: the Flagged Data Dashboard, the Data Visualization Dashboard, and the Administration Dashboard. In the following subsections we will detail the intended design of the QC Application as initially specified with the use of various mockups. For an exploration of the final design as was implemented see section 5.6.2.

5.5.1 Flags Interface

After users log in users this design detailed that users should be greeted with the page detailed in Figure 5.15. The first thing to note about this figure is the overall

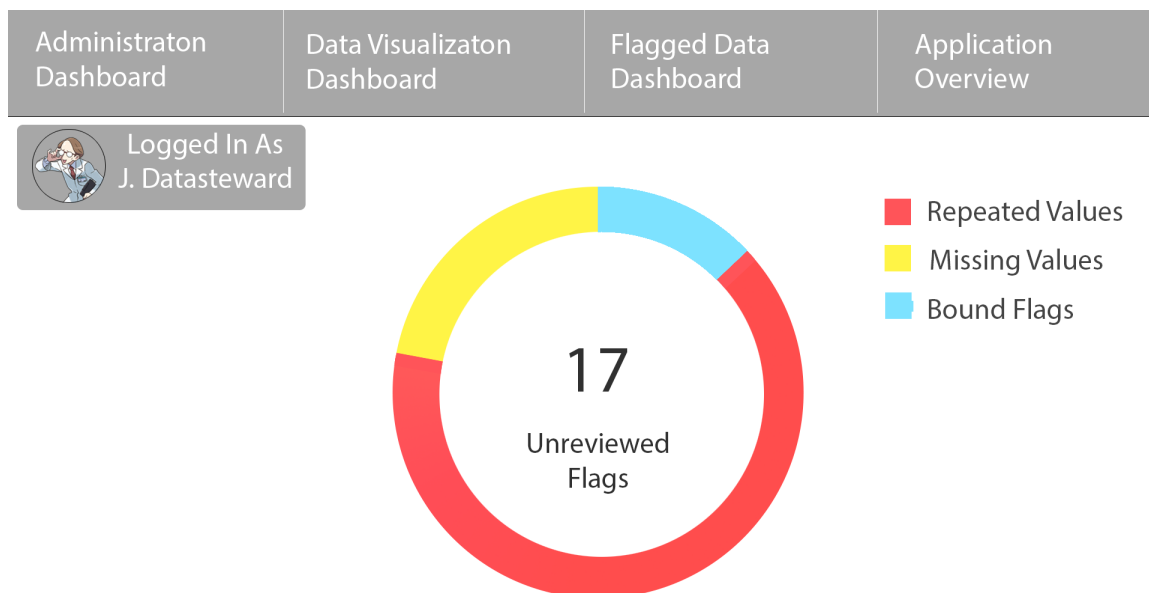


Figure 5.15: Main landing page for the Quality Control Application. It is the flagged data dashboard and it shows at a glance the number of flagged measurements in need of expert review. Other pages can be navigated to via the horizontal navigation bar on top.

structure of the webpage, the navigation bar is horizontally oriented like folder tabs and there is a lot of white space. The tab like navigation bar enables a cleaner design as large visual entities like charts are more easily aligned in the center of the page. This ensures that data visualizations, like the flag related doughnut chart are given a place of importance in the center. Building upon this, the UI utilizes white space and color to draw the eye to critical information and mitigate any distraction when evaluating important data.

Continuing with Figure 5.16 note the content of this page itself. From the prior image you will see that the a cursor is now present. Upon hovering over a portion of the doughnut chart the content in the center will change to give more detailed data reflecting the portion being interacted with. Additionally, by clicking this arc the user will be sent to the next page giving an even more detailed breakdown of flagged data of this type organized by data stream, visible in Figure 5.17. From this page a user can click on one of the flag bundles and pass forward into the Data Visualization

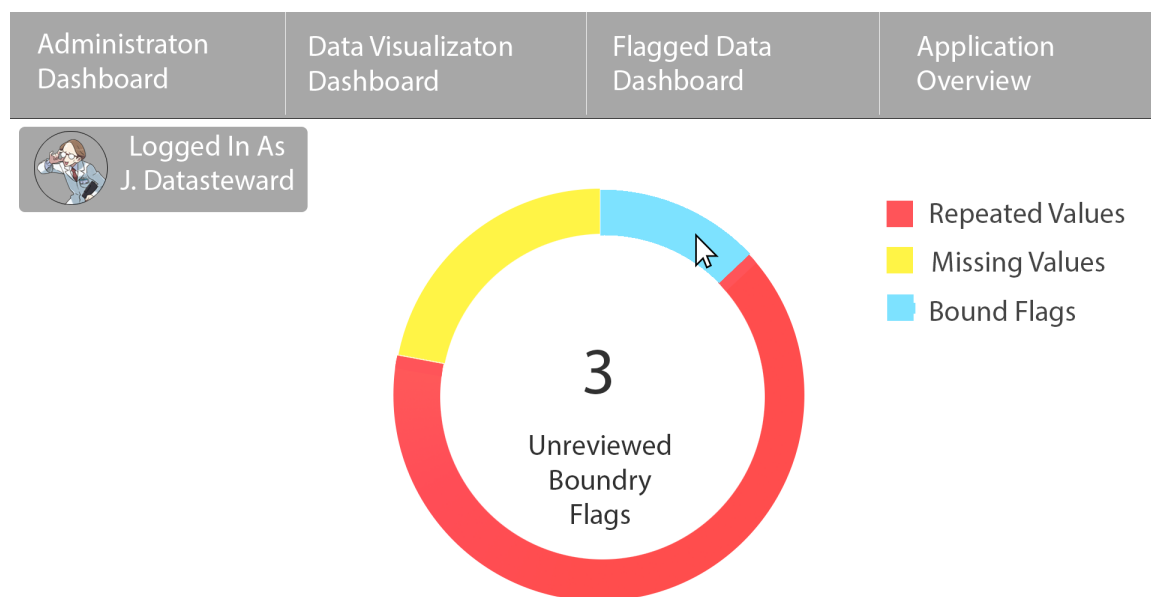


Figure 5.16: Main landing page for NRDC QC Application. Note the changed data in the center of the doughnut chart as the cursor hovers over an arc representing a subset of flagged measurements.

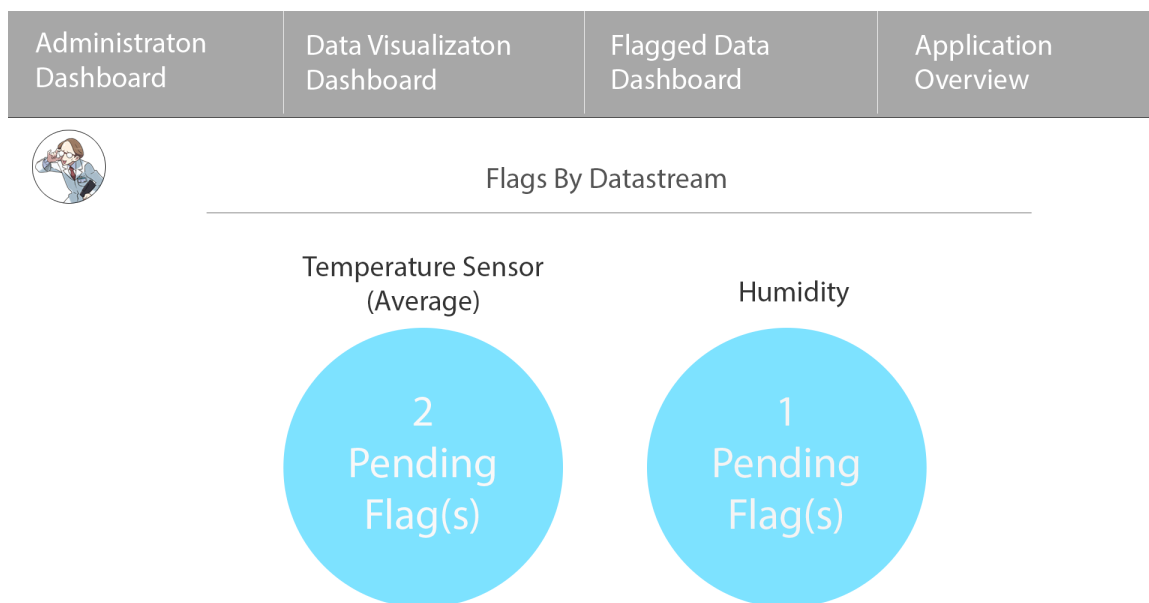


Figure 5.17: Flags of a certain type organized by datastream. Either of these flag bundles, represented by blue circles, are active links to the data visualization page where the associated datastream will be automatically visualized.

Dashboard with the data from that stream loaded into the line chart. This will be the primary way that data scientists get from the Flagged Data Dashboard to the Data Visualization Dashboard.

5.5.2 Data Visualization Interface

Whether the user navigated here via the link in the navigation bar or via the “Flagged Data Dashboard” workflow, the user will see the page detailed in Fig 5.18. However, they will only see the visualized data shown here, if they came via the Flagged Data Dashboard. If they came from the navigation bar link they will need to select the “Add Datastream” button from the left toolbar and choose a Datastream to be visualized before they see a line on the chart.

In this figure flagged data is rendered as red, while unflagged data is rendered in blue. This sort of color coding provides data stewards with at a glance information about Data Quality over time. It can help significantly when coordinating site visits for Quality Assurance and Maintenance because a Data Steward can, at a glance, see

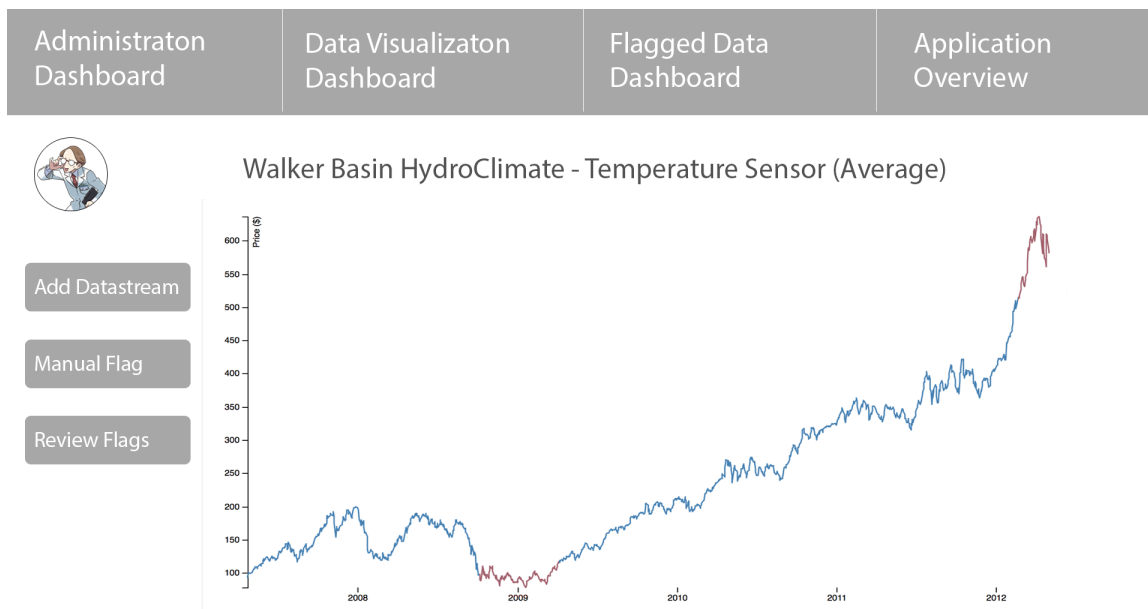


Figure 5.18: The main data visualization page. A toolbar on the left hand side of the screen provided various functionality to the user. They can add a datastream to the visualization, manually flag datapoints in the graph by slicing areas, or get a detailed list of all flagged datapoints associated with this stream. Flagged data is colored red in the visualization.

the relative ability of a sensor to log valid data over time. If there is a lot of red on a graph like this, it might be worth checking out the physical sensor.

For additional information, a user can hover over the chart at any point and get a quick breakdown of the measurement rendered at that point. This functionality is visible in Figure 5.19. The small text box displays relevant info about this Measurement like the value and the type of flag. Eventually, users will be able to click this data point to get an even more detailed breakdown of relevant flag related metadata, like information about the test that it failed or when it was flagged.

5.5.3 Administrative Interface

As configuration and administration of the Quality Control tests is such a crucial element of proper automated QC a substantial portion of the interface is dedicated to Administration. In Figure 5.20 users are presented with two simple and large

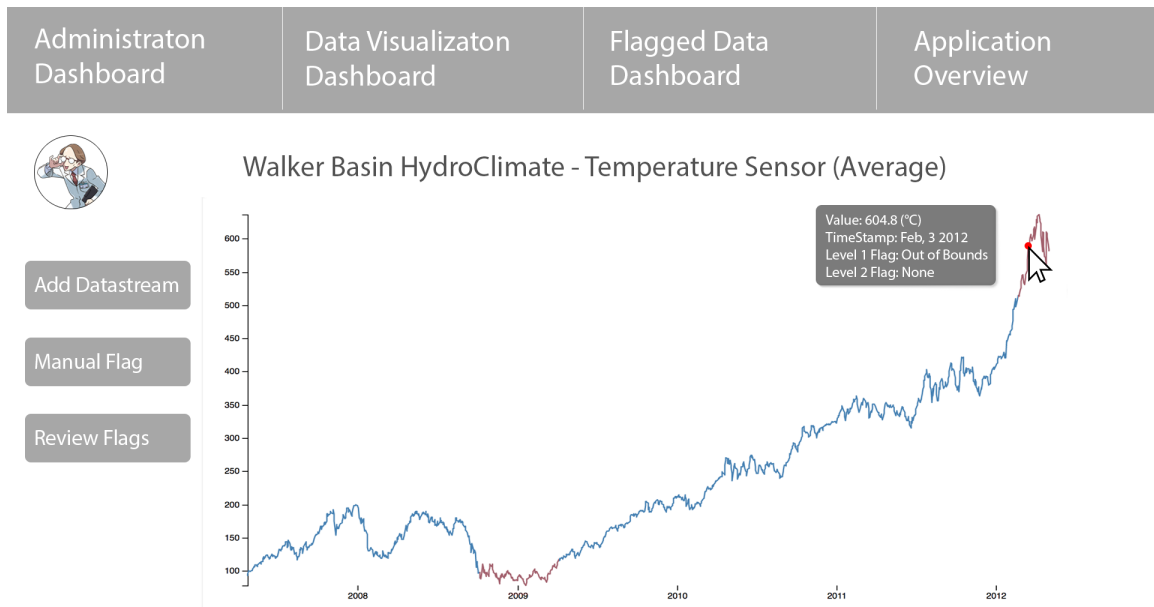


Figure 5.19: Data visualization page showing interactivity of graph. Even non-flagged data can be viewed by hovering in this manner.

buttons indicating possible paths of test related interaction and configuration. If an user wishes to view tests, they will be linked to a navigation page. Upon selecting a datastream they will view a listing of all tests associated with that datastream. From there they can reconfigure and adjust tests where desired.

If they choose to create a test, users will be taken to the view depicted in Figure 5.21. From here users can find a deployment by clicking the button in Step 1. From there they can selected an associated Datastream in Step 2 by clicking on the dropdown menu. The process continues with additional steps off of the depicted screen. In subsequent steps the user can select the type of test and any custom parameters that are required. From there they will be given the option to save their new test and upload it to the server to go into production by the next automated Quality Control Cycle.

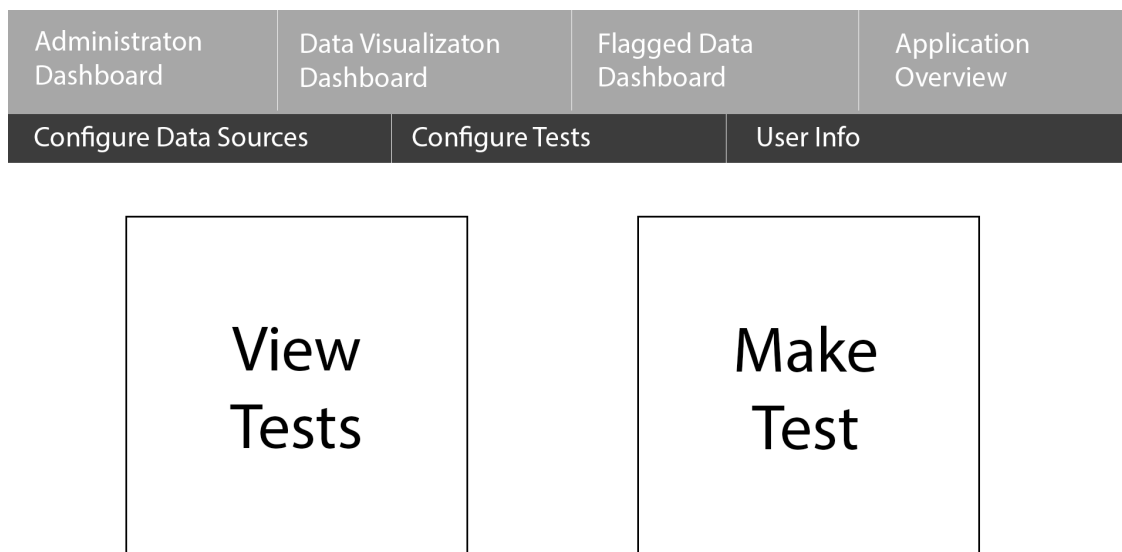


Figure 5.20: The configure tests screen. The user can navigate to a datastream to configure existing tests via the “view tests” button or they can construct an entirely new test by selecting “New Test”.

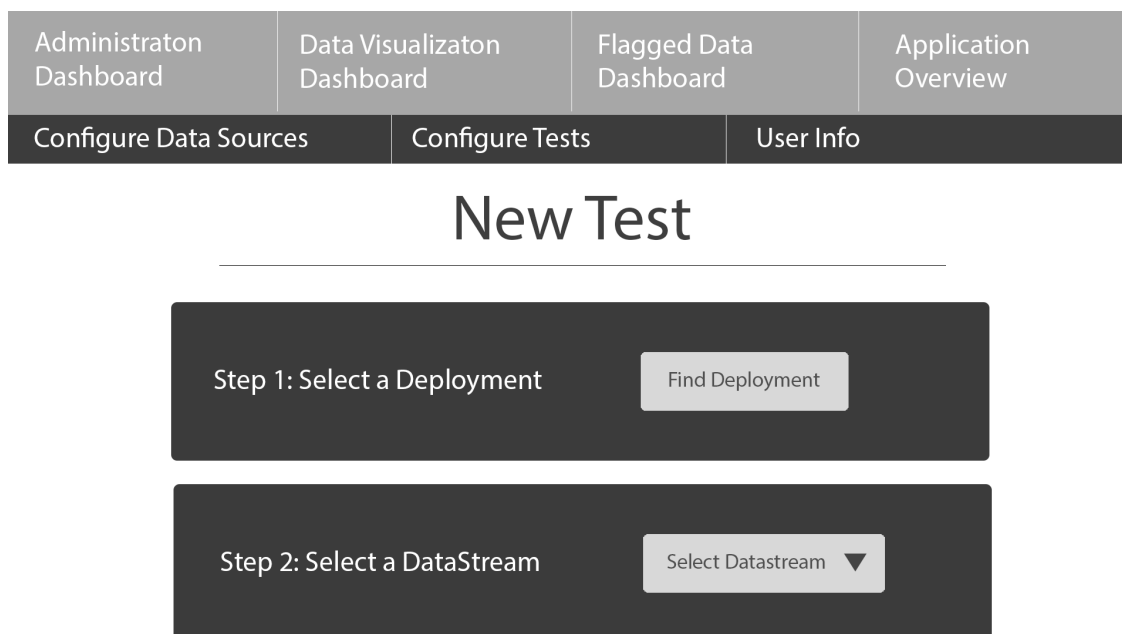


Figure 5.21: The new test creation page. Users are guided in creating a new test in a streamlined and step-by-step process.

5.6 Implementation Details

Following this comprehensive design process, a prototype application was developed to work with the NRDC and integrate into its existing data workflows. Although not fully featured to meet the specifications of the above specification and design details the developed application met a number of key functional requirements and proved the basic workflow of configuring and working with such an application. The following sections detail the narrative of this development and specify what was developed.

5.6.1 Backend

For the prototype of this application, five microservices were implemented across two phases: the Configuration Manager, Data Manager, Testing/Flagging Service, Flag Handler, and DataVis Handler. In addition to these microservices, all the classes except for a login class detailed in Section 5.3.2 were implemented to drive these microservices. These specific components were chosen because they best represented the core workflow of the NRAQC system and would be very likely re-integrated in future iterations of this software, if developed properly the first time.

In the primary phase of development four of the aforementioned microservices were developed as a series of Flask services that called upon the various classes defined in Section 5.3.2, implemented in Python, which provided the data retrieval and testing functionality. Using the methods in these classes, the Data Manager derived configuration information from a hard-coded XML file that allowed it to connect to the specified data source: the static NRDC Test Database hosted on a secondary server (to avoid critical failures which could compromise the original data).

After setting up a functional writing and querying service, the Testing/Flagging service followed. Again receiving functionality from the same set of classes, this service ingested a second hard-coded configuration XML file containing testing data. Using the ingested testing data the Testing/Flagging service creates all tests dynamically as “Test” objects and categorizes them by data stream so that they will always test

the correct measurement set. Upon successfully linking these two services together and verifying that they correctly retrieved, flagged and wrote back flagged data, development began on the client side and two edge services required to retrieve data from the data source.

In the second phase of development, which began a few months after completion of the initial phase, configuration classes and user interfaces were developed to meet further requirements specified in by the specification document. The development and completion of these classes enabled the existing NRAQC services to eschew use of the prior hard coded file used to define tests and now run tests however they were defined from the user interface. Development of the configuration classes was a non-trivial phase that required multiple redesigns of prior configuration files and classes which loaded and parsed them. Furthermore, as detailed in the User Interface subsection of this section, the implementation of a configuration service and class significantly changed the intended designs of these interfaces as specified in section 5.5. Despite attempts to implement the specified Data Source Configuration classes, due to constrictions of time and design problems they were defied implementation in this phase.

5.6.2 User Interface

Development on the UI was defined significantly by the efforts were made to mirror the initial UI designs. The results of these efforts were mixed but generally promising. Starting with the initial “landing” page of the NRAQC web interface, visible in Figure 5.22, the doughnut chart was relatively simple to implement out-of-box. Using the Flag Handler interface to pass down aggregated and formatted flag metadata from the NRDC test database, this doughnut chart required only a few lines of setup and worked exactly as designed. Unfortunately the specific colors used in this chart library, ChartJS, were difficult to configure so the colors currently visible are pretty much fixed. A box containing informative text was added to explain how the doughnut chart should be interacted with.

Near Real Time Autonomous Quality Control Dashboard

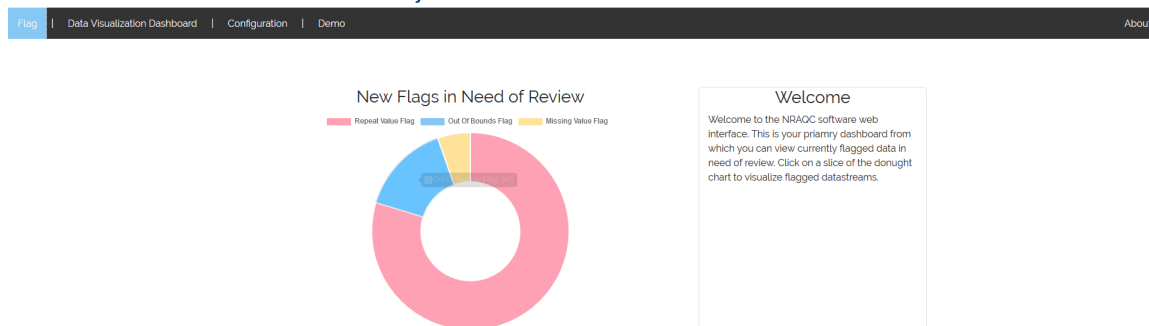


Figure 5.22: The main landing page of the NRAQC application.

By following the instructions in the welcome page and clicking on the doughnut chart, users are led to an intermediary page, Figure 5.23 where flags of the selected type are grouped by their specific datastream. These little bubbles are constructed using D3.js [16]. Although a color-coded through line was intended to link particular flags with particular colors, this had to be abandoned for the purposes of this application when configuring doughnut chart colors became increasingly difficult. On hover the numbers in these circles change color from white to black to provide user feedback.

By clicking on one of the circles here the user is lead to the core data visualization page of the NRAQC application, visible in Figure 5.24. On this page, the user has multiple options but the primary focus lies on the chart taking up more than half the screen.

Although visually pleasing, this chart lacks functionality intended from original designs of this software. The user can hover over specific data points in this chart (indicated by the red circles on our red line), however the data popup which results has very limited information. Specifically, it specifies the x and y values and nothing more. Despite making numerous attempts, with multiple visualization libraries, to

Near Real Time Autonomous Quality Control Dashboard

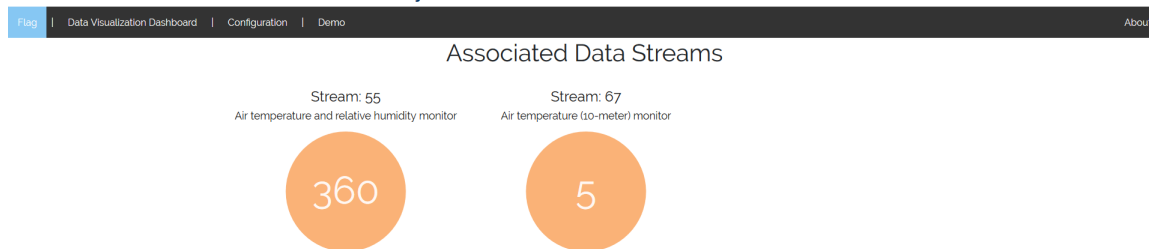


Figure 5.23: A page which organizes flagged data points by the stream they were flagged on. On this page we can see that there are 365 missing value flags, with 360 originating from one datastream and 5 from another.

Near Real Time Autonomous Quality Control Dashboard

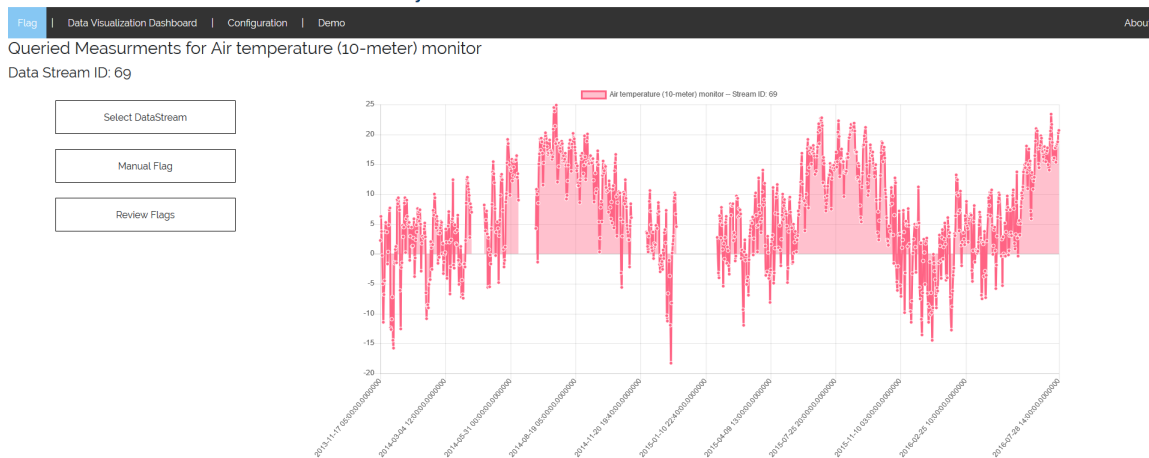


Figure 5.24: An image of the main data visualization page for the NRAQC software. This datastream shows the results of inserting empty spaces to represent gaps in a dataset.

affix additional metadata to this tooltip popup no solution was found which did not require significant fundamental development. Additionally, there existed no simple way to specify that certain values should color the lines on the chart a certain way so the chart is one color.

In order to address this lacking functionality on the chart, a second interface was developed which users could switch between to view flagged datapoints as a list. Visible in Figure 5.25, this interface lists the flag, value, and timestamp of each flagged datapoint in the current visualized datastream. To indicate how such a interface could be used if more richly developed, a text box for data steward annotations was added to the right of each flag. Beyond this textbox, no manual flagging functionality was developed for this prototype.

Moving from the flag review and visualization pages, a series of pages were developed to facilitate the easy configuration of tests and affix them to the proper datastreams. Using a navigation workflow demonstrated with Figures 5.26-5.29, users navigate down a hierarchy of metadata which specifies the location of certain instrument deployments. Along the left had side the “Nav-History” box specifies the prior

Near Real Time Autonomous Quality Control Dashboard

Flag | Data Visualization Dashboard | Configuration | Demo About

Queried Measurements for Air temperature (10-meter) monitor
Data Stream ID: 69

Value	Time Stamp	Flag	Annotation
None	2014-04-17 21:00:00.0000000	Missing Value	<input type="text"/>
None	2014-04-17 21:10:00.0000000	Missing Value	<input type="text"/>
None	2014-04-17 21:20:00.0000000	Missing Value	<input type="text"/>
None	2014-04-17 21:30:00.0000000	Missing Value	<input type="text"/>
None	2014-04-17 21:40:00.0000000	Missing Value	<input type="text"/>
None	2014-04-17 21:50:00.0000000	Missing Value	<input type="text"/>
None	2014-04-17 22:00:00.0000000	Missing Value	<input type="text"/>

Left sidebar controls:

- Select DataStream
- Manual Flag
- Review Flags
- View Chart

Figure 5.25: A list of all flags associated with the currently visualized datastream, with their timestamps and the name of the flag.

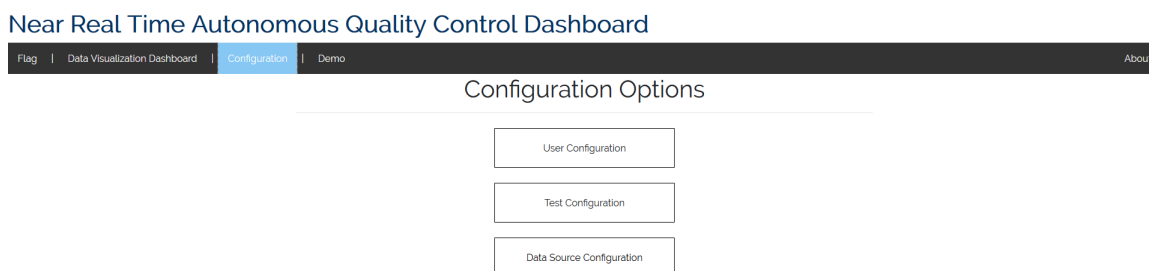


Figure 5.26: The top level options for configuration. User and data source configuration were not implemented for this prototype.

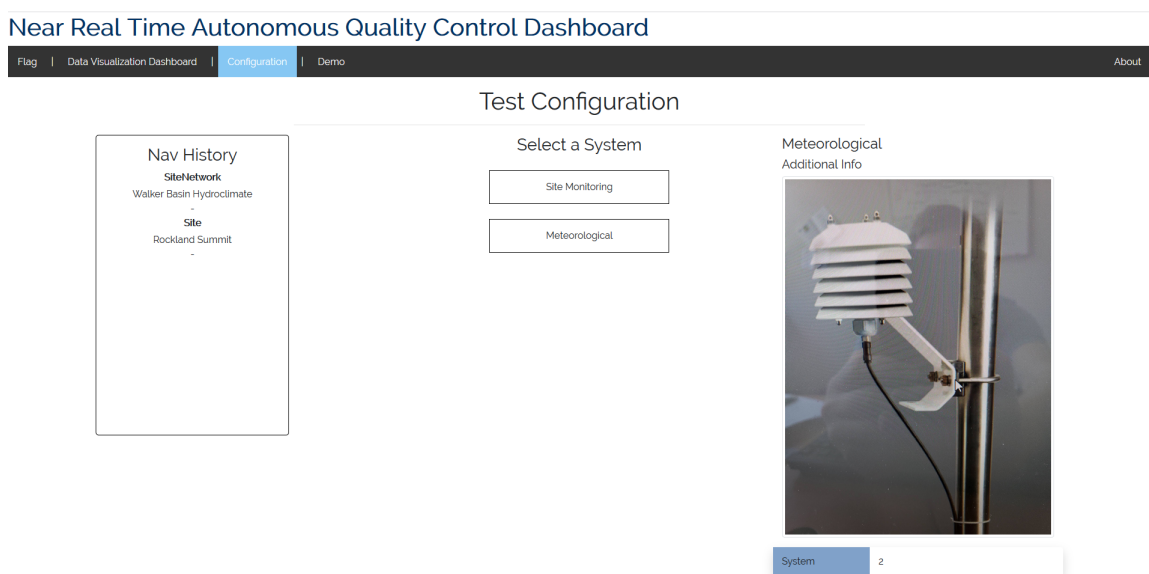


Figure 5.27: A snapshot of the hierarchical navigation used to find a particular datastream which can be tested. The image on the right was uploaded from the Quality Assurance Application detailed in Chapter 4. On the left, the nav history pane shows breadcrumbs of where we have come from. We can click on any of those breadcrumbs to return to a prior page.

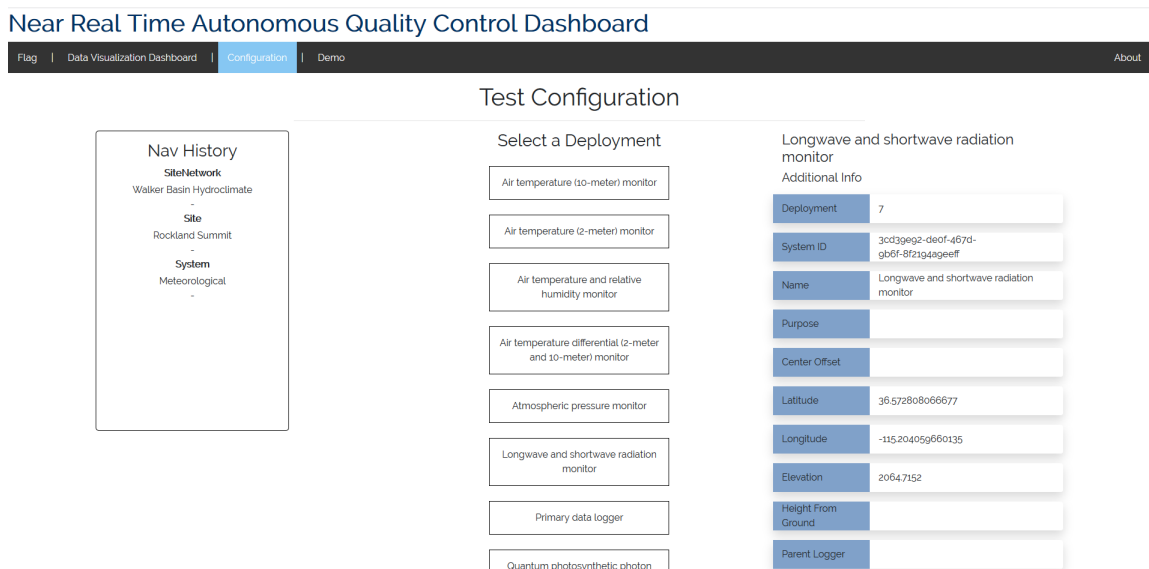


Figure 5.28: A snapshot showing the variety of metadata we can retrieve from the QA app and display in the QC app.

pages which users have navigated down. On the right hand side, the list of boxes contains metadata about the current level our user has navigated to, including images. For data curators familiar with the associated QA application this interface should feel logical and provides an example of how the prototype NRAQC application benefits from metadata collected by the QA application. Although, somewhat trivial in its current implementation, this synergy will be explored in greater detail in Chapter 7.

Once users have navigated to their intended datastream, users are given the option to create new tests or modify existing tests. In Figure 5.29, a datastream has been chosen which has a pre-existing test and a new test. If the user clicks on the new test, “Create a New Bounds Test”, the page in Figure 5.30 appears. The content on this page is dynamic and provides a number of fields specific to the type of test being run. For “bounds” tests it is two, for repeat values it is only one, a threshold.

When the user inputs information into these blank forms, a “save changes” prompt appears, visible in Figure 5.31. Upon clicking this button to save, the user receives feedback in the form of a green banner along the bottom of this page. This

Nav History

- SiteNetwork
 - Walker Basin Hydroclimate
- Site
 - Rockland Summit
- System
 - Meteorological
- Deployment
 - Air temperature and relative humidity monitor
- Datastream
 - Relative humidity Maximum -- 00:10:00.0000000

Create a New Test

Create New Bounds Test

Select an existing Test

Repeat Value

Relative humidity Maximum -- 00:10:00.0000000

Additional Info

Category ID	2
Data Interval	00:10:00.0000000
Data Type Name	Maximum
Deployment ID	353fb39b-11ba-4d1a-8d03-24f6809d1c5
Deployment Name	Air temperature and relative humidity monitor
Ended	
Property ID	249
Property Name	Relative humidity
Started	2013-11-13 00:00:00.0000000
Stream ID	54

Figure 5.29: A snapshot showing our options for creating new tests on a chosen datastream. On this datastream we already have an existing test for repeat values we can modify and we also can create a new test which checks for values which exceed desired ranges of our measurement series.

Nav History

- SiteNetwork
 - Walker Basin Hydroclimate
- Site
 - Rockland Summit
- System
 - Meteorological
- Deployment
 - Air temperature and relative humidity monitor
- Datastream
 - Relative humidity Maximum -- 00:10:00.0000000

Test Configuration

Bounds Test

Test Parameter Name

Current Parameter Value

New Parameter Value

Min

Max

Edit

Return to Previous Screen

Figure 5.30: An example of a test configuration page. This particular test checks to verify that a datapoint does not exceed either the minimum or maximum range specified in these form fields.

Near Real Time Autonomous Quality Control Dashboard

The screenshot displays the 'Test Configuration' page. On the left is a 'Nav History' sidebar with a tree structure:

- SiteNetwork
 - Walker Basin Hydroclimate
 - Site
 - Rockland Summit
 - System
 - Meteorological
 - Deployment
 - Air temperature and relative humidity monitor
 - Datastream
 - Relative humidity Maximum --
 - 00:10:00.00000000
 -

The main content area is titled 'Bounds Test' and contains the following table:

| Test Parameter Name | Current Parameter Value | New Parameter Value |
|---------------------|-------------------------|----------------------------------|
| Max | 40.8 | <input type="text" value="100"/> |
| Min | -40.1 | <input type="text" value="10"/> |

Below the table are two buttons: 'Undo Changes' and 'Save Changes'. An 'Edit' button is located in the top right corner of the configuration area.

Figure 5.31: When information has been input into a form field the save changes button appears.

can be seen in Figure 5.32. Editing an existing test follows almost exactly the same workflow.

Finally, in order to demonstrate the functionality of this software on a static database, which does not receive any new inputs like the live database would, a final interface was implemented called Demo. Like the name implies, this interface demonstrates the functionality of this application by performing batch tests on data in the NRDC database. As these batch tests could be very time consuming, due to the nature of how voluminous the tested data is, the Demo Service was multithreaded so that it could report back its progress to the demo interface. In Figure 5.33 you can see an example of one stage of the Demo interface reporting on its progress.

Near Real Time Autonomous Quality Control Dashboard

Flag | Data Visualization Dashboard | Configuration | Demo | About

Test Configuration

Nav History

- SiteNetwork**
- Walker Basin Hydroclimate
-
- Site**
- Rockland Summit
-
- System**
- Meteorological
-
- Deployment**
- Air temperature and relative humidity monitor
-
- Datastream**
- Relative humidity Maximum --
- 00:10:00.00000000
-

Bounds Test


| Test Parameter Name | Current Parameter Value |
|---------------------|-------------------------|
| Min | -40.1 |
| Max | 40.8 |

Your test parameters were successfully modified and saved on the server.

Figure 5.32: When changes are saved the user is notified via a green strip at the bottom of this form.

Near Real Time Autonomous Quality Control Dashboard

Flag | Data Visualization Dashboard | Configuration | Demo | About



Running Tests.
1 Datastreams
out of 6

Figure 5.33: The Demo page of the NRAQC app. This page notifies users about the current status of the demo via helpful text updates.

6 Data Repair

6.1 Role in Keystone

The Improved Robust and Sparse Fuzzy K-Means (iRSFKM) software detailed in this chapter fulfils the intended purpose of the Data Repair module in Keystone through the rapid and accurate imputation functionality it provides. The final requirement to get our Raw Data to a finished data product is that it be complete as possible. The Data Repair module of our model suggests that completeness of a data product can be fulfilled semi-artificially by making accurate statistical guesses about what values should go in established holes in our dataset.

As has been loosely established in Chapter 2, many options exist for data imputation. For our application of the Data Repair portion of the Keystone model we implemented a modified fuzzy k-means algorithm and ran it on multiple GPUs. This implementation repairs real-earth science data accurately and quickly and can be leveraged in an integrated pipeline to help the NRDC produce a more complete and accurate data product that reduces the time-to-science for researchers.

6.2 Implementation

Four iterations of the improved RSFKM algorithm were implemented for experimentation and analysis. First, a standard CPU implementation of the RSFKM was implemented using a generic solver for Equation (2.3). This implementation was developed to provide a clear baseline of the existing algorithm for timing purposes. Next, the sequential CPU-only implementation was optimized with the introduction of library

free convex optimizer in lieu of the generic solver. From there, a single GPU implementation was introduced and optimized for speed and overhead efficiency. Finally, to account for restrictions on GPU memory and produce more consistent results, a multi-GPU iteration of this software was implemented.

6.2.1 Convex Optimization

In the work by Xu, *et al.*, the authors reference the use of “the technique” utilized by Huang, *etx. al.* [51] to solve Equation (2.3). To speed up development time for this baseline code and to promote reproducibility among computer scientists who are not intimately familiar with convex optimization, we diverged from the method in Xu, *et al.* and utilized ECOS convex optimization solver[30] through the CVXpy interface [26]. Unfortunately, the introduction of this generic solver caused problems for plans of a GPU adaptation of this algorithm.

Presently, no generic solver currently exists which leverages the power of GPUs to speed up its numerous and dense calculations. Since preliminary timings of this solver indicated that this was a substantial bottleneck in the RSFKM algorithm, a workaround was necessary to enable the use of GPU architectures with RSFKM. To solve this problem, a library free solver, comprised of optimized C code generated by CVXGEN [64], was integrated into a modified sequential implementation of RSFKM. This improved implementation was significantly faster and provides a best case baseline to measure GPU versions against.

The code generated by CVXGEN was much more sympathetic to a GPU implementation of RSFKM compared to CVXpy as it was library free and written in C, which directly maps to CUDA. However, this code also introduced new problems hindering a successful GPU implementation. First, it generates code that only works on a pre-defined, constant vector size. In our case, that means it only works for a fixed number of centroids. Next, as this generated code was optimized for embedded systems, many frequently accessed variables were set at global scope. As this code would have to be adapted to run sequentially on a single thread (but with multi-

ple instances running in parallel across multiple cores) this scoping creates problems. CUDA doesn't have an equivalent global scope that exists independently for each thread, so a means to trick the architecture was required.

6.2.2 GPU Adaptation

As alluded to in Section 2.6.2, the approach of prioritizing the adaptation of optimization code to GPUs came from two directions. First, the mathematical minimization formula for membership is fundamentally independent, indicating that calculations for membership vector u_i can occur at the same time as u_{i+1} . This makes the corresponding algorithm embarrassingly parallel. Second, the significant bottleneck of this minimization problem demanded resolution before other approaches to optimization and parallelization could be considered.

The problem of the globally scoped variables was solved first. To properly adapt the generated C code, a method was devised to maintain the structure of the program as developed to run on a CPU. To maintain the illusion of global scope, per-thread, the entirety of generated code, approximately 1,500 lines, was encapsulated into a class-like struct. Encasing our C code into a struct allowed all required variables to be treated like private members which could be freely and safely accessed by the component functions which performed the minimization solving. The struct functions themselves were labeled as `__device__` functions and could then be called freely by individual threads from the "Update Membership" kernel.

As can be seen in Figure 6.1, the modified kernel was called on n blocks, each using one thread. Although this configuration seems naive, attempts to call this sequential code with multiple threads per block, in multiples of 32 to take advantage of warp efficiency, saw much slower results than the one-thread-per-block approach. Furthermore, very few threads-per-block could be instantiated due to a lack of register space. This is likely due to the size of the solver struct being loaded to run on the GPU. By using this structure, our solver could make the best use of GPU resources to perform the embarrassingly parallel computations promised by Equation (2.3) in

the most straightforward manner possible.

6.2.3 Further Optimization

After converting the primary bottleneck of the minimization solver into GPU code, many other opportunities for optimization became evident. In addition to the aforementioned “Update Membership” function, there existed three other computationally intensive functions which comprised the core functionality of this algorithm. Those functions are **build_h_matrix**, **find_centroids**, **update_S**. Generally, these functions involved some kind of matrix manipulation with a sum reduction and could be easily mapped one matrix cell to a block with multiple threads doing reduction and simple addition work in parallel. Shared memory was used for each of these functions as a buffer to hold summed data, pre-reduction, for fast access and simplicity of implementation.

Through rewriting these functions in CUDA to run on the GPU, and adding supplemental functions to store and calculate auxiliary scalars between iterations, the entirety of this program (within a single iteration) was successfully configured to run on the GPU. We successfully minimized overhead with this full conversion by initializing all derived matrices, **U**, **V**, **H**, and **S** in global memory on the GPU. Each of these matrices can be maintained on the GPU’s main memory between the core kernel calls which perform the computations of our clustering algorithm. Although highly efficient, this memory configuration caused problems in terms of space efficiency which are addressed by the multi-GPU implementation of this algorithm.

The only matrix which could not be initialized on the GPU was the original data matrix **X**, because its contents are read from the disk. This however does not significantly impact overall run time because the overhead of transferring this data to the GPU occurs only once before clustering begins. After this, the only continuous data transfer occurs at the end of each iteration when our centroids, **V**, are transferred back down to the CPU to check for convergence. The overhead of this transfer is very minimal however, as the number of centroids is generally going

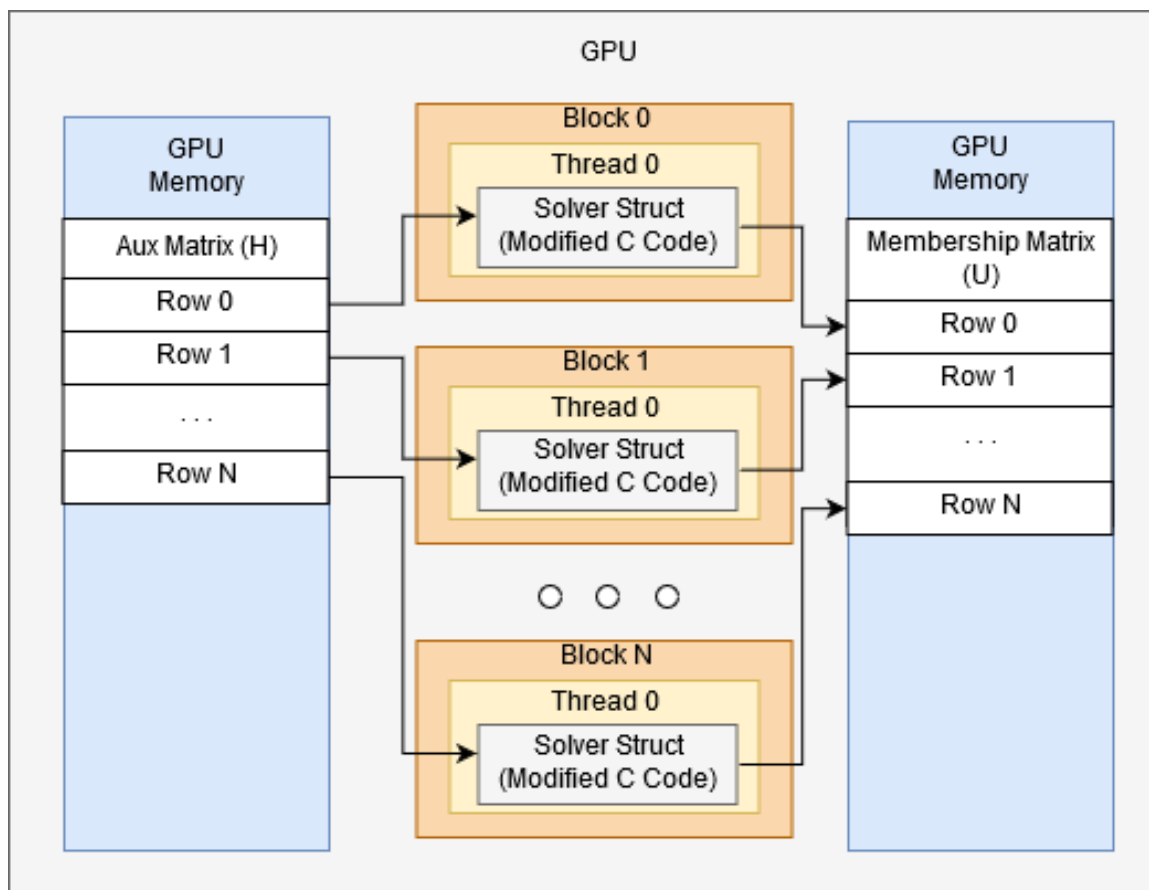


Figure 6.1: A diagram showing a basic breakdown of the implementation and operation of the core solver implemented as a CUDA struct from modified C code. This diagram shows the mapping of the solver to each block and indicates how the problem is decomposed, where each line in our membership matrix is solved in parallel on the GPU. The decision to call this solver kernel on one thread per block came from slowdown observed with more than one thread per block.

to be relatively small to provide meaningful clustering. The final overhead from data movement between host and device comes from the output of the membership matrix U . Like our data matrix, this matrix transmits between GPU to CPU only one time after all the iterations of this clustering algorithm have concluded. Because of this, the overhead of this retrieval can be considered negligible.

6.2.4 Multi-GPU

The need for a multi-GPU implementation of this algorithm was driven by the memory limitations of our single-GPU implementation. `calculate_centroids` requires a substantial sum reduction resulting from a matrix multiplication between the data matrix and an auxiliary matrix which holds a regulating scalar. After multiplication, but before reduction, result vectors are stored in shared memory assigned to a block. This requires our shared buffer to hold n spaces of size 8 bytes for the double precision values. CUDA provides a max of 48 KB per block. Accordingly, this only allows for approximately 6,000 double precision values to be processed by this function. At one point in this algorithm, this function handles every data value we intend to cluster. With the space limitations, this indicates that we cannot cluster more than 6,000 values with this program.

Although there are certainly many approaches and solutions to this problem, we chose to leverage data decomposition and split our calculations across multiple GPUs. By randomly sampling data points and distributing them across GPUs we can accurately cluster up to 48,000 data points in the same amount of time it would take for 6000. Although the memory limitation remains, per-gpu, it becomes significantly less impactful with each GPU we add. This horizontal scaling is not significantly impacted by overhead due to the design of our per-gpu algorithm, where all substantial matrices are instantiated and maintained on each GPU's main memory. Data transfer occurs minimally in between interactions to check for convergence and at the end to transfer back results for imputation.

6.2.5 Data Imputation

The data imputation algorithm was implemented in Python as a collection of methods which performed basic pre-processing, called the above detailed GPU clustering algorithm, removed data for imputation and performed the imputation algorithm specified in Equation (2.5) using the centroids and returned membership matrix retrieved from the GPU. The implemented imputation method checked its accuracy using a simple RMSE algorithm finding the difference between the actual values removed from the dataset earlier against those which were imputed.

7 Validation and Discussion

7.1 Clustering Experiments

To evaluate the improved RSFKM algorithm on a single GPU and multiple GPUs, timings were collected on repeated clustering of a set of environmental sensor data downloaded from the Nevada Research Data Center’s website [77]. Preliminary experiments showed that changing the number of features on our data and adjusting the number of centroids did very little to influence overall runtimes for either CPU or GPU implementations of this algorithm. So, for all the following experiments, the number of centroids used is fixed at 15 and the number of features is fixed at 11.

Table 7.1: An excerpt of environmental data used for bench marking the effectiveness of the improved RSFKM clustering algorithm. The values depicted here are humidity and temperature measurements collected from many sites across Nevada.

| Timestamp | Temp. (C) | Temp. (C) | Temp. (C) | Temp. (C) | Humid. (%) |
|----------------|-----------|-----------|-----------|-----------|------------|
| 1/25/2018 1:19 | -5.678 | -6.482 | -6.499 | -6.455 | 0.491 |
| 1/25/2018 1:20 | -5.654 | -6.474 | -6.499 | -6.452 | 0.492 |
| 1/25/2018 1:21 | -5.697 | -6.462 | -6.479 | -6.452 | 0.494 |
| 1/25/2018 1:22 | -5.774 | -6.481 | -6.499 | -6.449 | 0.494 |
| 1/25/2018 1:23 | -5.788 | -6.491 | -6.515 | -6.472 | 0.494 |
| 1/25/2018 1:24 | -5.793 | -6.503 | -6.519 | -6.478 | 0.492 |
| 1/25/2018 1:25 | -5.732 | -6.515 | -6.538 | -6.492 | 0.494 |

7.1.1 Experimental Setup

The following experiments were performed on a remote server containing 24 2.00 Ghz Intel Xeon CPUs connected over two PCIe buses to 8 GeForce GTX 1080 GPUs. The GTX 1080s are grouped 4 cards to a single bus. Each GPU has an available memory of 8GB with 6KB of shared memory available per block. Each GPU implements the Pascal architecture which provides support for advanced processing features like unified memory.

As an initial proof of concept, the clustered data set was a selection of 550,000 data points of time series data organized into 50,000 vectors with 11 features. Each vector represented a per-minute log of autonomously collected meteorological data with each feature representing a measurement collected at that minute from a distinct sensor. Table 7.1 shows an excerpt of the data set used.

7.1.2 Data Imputation

Although not explored in great detail for this paper due to constraints of space, this algorithm performed well in providing reasonably accurate data imputation. With RMSE values as low as 0.18 the GPU implementation of this algorithm provided accurate clustering on par with the CPU implementation. Since the applications and datasets were very different between this paper and those tested by Xu, *et al.* [110], a more detailed imputation experiment will be required and explored in future work.

7.1.3 Timings

Fuzzy K Means clustering, is a variable length iterative algorithm that does not converge uniformly at a set number of iterations. Between runs where the same data set is organized into the same number of clusters, convergence can occur in half as many iterations as expected. Accordingly, this makes collecting timings of full program runs problematic, because times can vary wildly. To solve this problem the runtimes collected and manipulated in the following sections were calculated per-iteration by dividing the overall runtime of this algorithm by the number of iterations

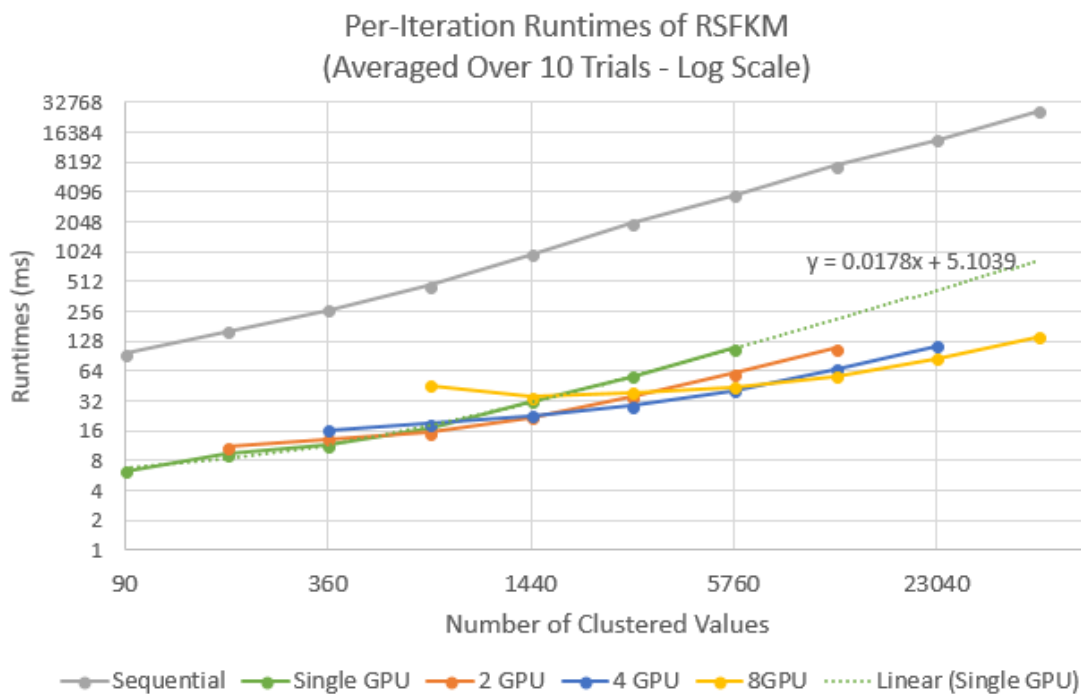


Figure 7.1: A line plot showing the runtimes of the RSFKM algorithm running sequentially on a CPU and in parallel on one, two, four and eight GPUs. The runtimes are per iteration of the clustering algorithm and averaged over ten trials. The dotted line indicates a trendline which shows the projected runtimes of a single GPU experiment. The equation next to the trendline was used to derive runtime values for efficiency calculations.

taken. Using this timing scheme significantly reduced outliers in our timing data and produced meaningful results.

The raw per-iteration timings, shown in Figure 7.1, show a comparison between the sequential implementation of RSFKM and our modified GPU implementation, on one, two, four and eight GPUs. The sequential runtimes, indicated by the grey line, indicate a clear linear curve which is to be expected by the $O(NVF)$ complexity of our algorithm. As V , the number of centroids, and F , the number of features in the data vectors, are fixed at relatively small sizes compared to N , the algorithm becomes linear as N becomes sufficiently large.

At a glance its evident that GPU implementations were very successful in reducing the runtimes compared to the sequential algorithm. On this logarithmically scaled

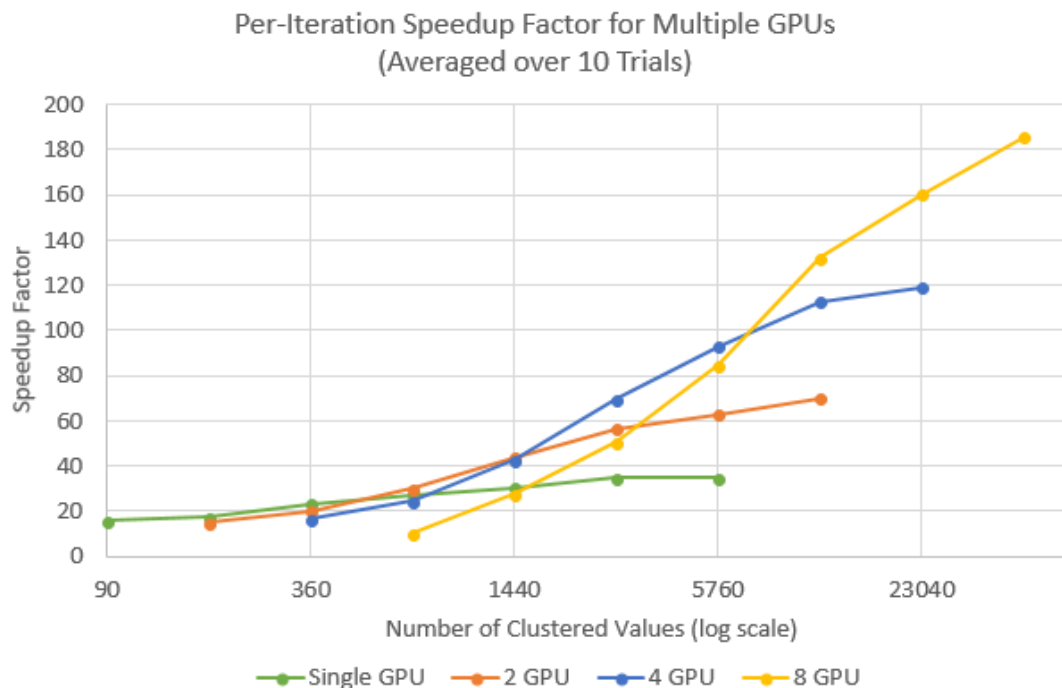


Figure 7.2: A line plot showing the per-iteration speedup factor of singular and multi-GPU experiments run with the RSFKM algorithm. For larger numbers of clustered values, the addition of more GPUs produces near equivalently scaled speedup factors.

graph, per-iteration runtimes of GPU implementations remain relatively level as data inputs scale from 90 vectors to approx. 40,000, ending with runtimes of 128ms up from 6ms. By comparison, across the same spread of processed data points the CPU implementation grows significantly in runtimes, from 128ms to almost 32000ms. The difference between these runtimes is so severe that when rendered with the CPU runtimes, without a logarithmic scaling, the GPU runtimes cannot be seen as anything other than a multicolored horizontal line on the bottom of the graph.

The significance of this difference in runtimes is further reinforced by Figure 7.2 which shows the speedup factor of various GPU timings when compared with the sequential timings. The speedup factor, S , was calculated with the following equation: $S = t_s/t_p$, with t_s and t_p representing sequential and GPU runtimes respectively. Overall, the speedup was very promising with a low of around 10x speedup for 8

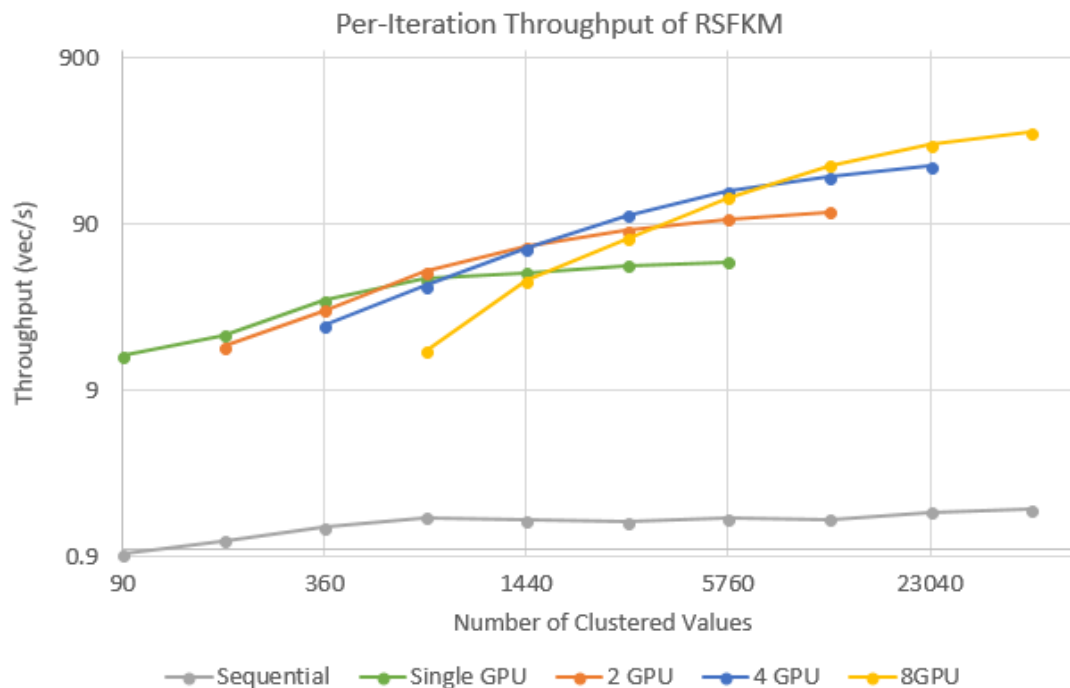


Figure 7.3: A line plot showing per-iteration throughput of the RSFKM algorithm as run on CPU, one, two, four and eight GPUs. Throughput for this paper was expressed in terms of vectors/ms and was calculated as $thp = n/ms$ where n is the number of vectors input for clustering. GPU throughput handily outstrips sequential throughput by a wide amount, even when processing a small number of values.

GPUs (with a small number of values) and a high of over 180x speedup for 8 GPUs. Speedup was shown to be more significant as more GPUs were added and additional data points could be processed. This increase in speedup is certainly consistent with expectations because there is very little cost to calculations performed with two GPUs, compared to one due to a lack of communication overhead. Broadly this indicates that when the fundamental overhead of CPU/GPU communication is overcome by data processing then speedup scales with the number of GPUs utilized.

Analysis of throughput and efficiency reinforce the conclusions made in reference to timings and speedup. In Figure 7.3, it's shown that overall throughput increases with more GPUs but only for sufficiently large amounts of data. For smaller amounts of clustered data we see a general dropoff of throughput when compared with fewer

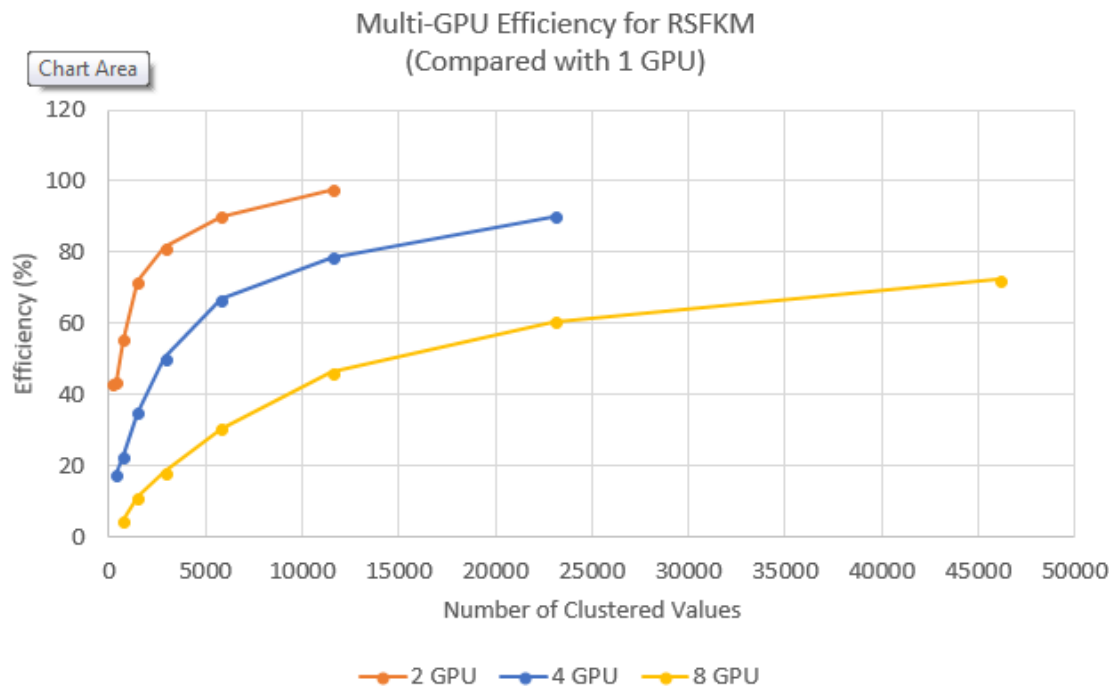


Figure 7.4: A line plot showing the relative efficiency of multi gpu implementations vs. single GPU implementation. Calculated as $E = t_{gpu} / (t_{gpuN} * N) * 100$, where N is the number of GPUs being used, this graph indicates how much meaningful work each GPU is performing.

GPU configurations, again likely due to increased overhead that comes with the addition of multiple GPUs.

Similarly, Figure 7.4 most clearly shows that at lower values very little work is being done on each GPU. As more GPUs are added the lack of work being done is exacerbated dramatically. For 360 values, 2 GPUs get near 40% efficiency with below 10% 8 GPUs. This difference in efficiency drops off dramatically however as more data is processed by each GPU. We do see that overall efficiency still drops off with the addition of more GPUs despite increased throughput and speedup. Again, this is likely due to increased competition for bus access which is harder to offset as more GPUs are added.

Within the GPU runtimes themselves some characteristics of these lines should be noted. First, there is a trendline on the single GPU runtimes. As explained in Chapter 6, space limitations prevented successful operation of this program on

one GPU with more than 6000 initial values. A trendline enables us to visualize and estimate runtimes going out towards the max processed 40,000 values. This trendline and its derived data will help provide efficiency data for our multi-GPU implementations.

It should be noted that with the two, four and eight GPU experiments in Figure 7.1 a bow-like curvature can be seen. This is almost certainly due to the overhead of transferred data between our multiple GPUs and the CPU. Since only two PCIe buses connect GPUs and CPUs the overhead of data transfer becomes more pronounced as GPUs are added and each GPU must compete to transfer data. And, even though there is relatively little data transferred between the GPU and CPU with this algorithm, there is certainly enough, especially with the per iteration transfer of our centroids, that slowdowns occur when there is not a sufficient amount of data for each GPU to process. The downward curve occurs as a result of more data being computed on each GPU and causing less frequent calls back to the CPU to check for convergence. This in turn, reduces simultaneous demand for the limited bus access.

7.2 QA Discussion

Lacking a formal user study, the validation of the proposed QA software is broken into a discussion about how QA has impacted the NRDC and the potential which exists for the QA application to affect the greater scientific community.

7.2.1 NRDC Impact

The successful implementation of the QA application changes the face of quality assurance in the field significantly for existing projects in the NRDC. The inclusion of a dedicated application impacts the workflow of sensor technicians and researchers by substantially augmenting current data management capabilities. Data stewards performing QA on sensor networks benefit from this application in several ways over traditional methods: uniform data entry, centralized QA data storage with synchro-

nization and a usable interface facilitating the utility of the above benefits.

The problem of uniform and accurate data entry naturally occurs in any system reliant upon human interaction as the primary interface between a means of measurement and the means of logging. This problem is further exacerbated when technicians are deployed to remote areas, often equipped with only a notebook. It can be very hard to meaningfully restrict metadata and service logging, as different people are going to include different data that they find relevant to a QA expedition. The use of form fields significantly normalizes data input by restricting users to only give information deemed necessary and sufficient to detail the quality assurance practices performed. An example of these forms can be seen in Figure 4.7. In the case of intrinsically non-structured data, the option to attach documents is provided. This enables a diversity of data input methods.

Previously, QA-relevant metadata collected and maintained by technicians was held in a decentralized heterogeneous collection of notebooks, spreadsheets and program comments. This proved problematic internally, as audits and reviews of quality assurance processes could not be effectively performed in a timely manner. Externally, this lack of centralized provenance-tracking metadata damages the integrity of collected data, as logs are not comprehensively tied to data streams. This limits the re-usability of collected data. With a central repository and dedicated backend infrastructure, the QA App significantly improves the maintenance of QA and metadata logs by providing a centralized, organized database to store this information and bind it to existing data streams. With the frontend mobile component automatically syncing with remote servers, users need not worry about any logistics of storing and formatting QA data for future use. They now only need to perform their normal maintenance and installation practices and fill out the form fields detailing their work.

7.2.2 Broader Impact

It has been well documented that an issue of paramount importance to the greater scientific community is the need for collected data to be accessible, findable, interoperable and reusable. [55, 107, 108]. As scientific research has become increasingly collaborative with the growth of internet technologies, the free and easy distribution of data across the internet has not kept pace [12, 81, 100]. While there are many factors that contribute to this data need/accessibility gap, one prominent problem is a lack of identifying metadata accompanying datasets [81, 108]. That is the goal of this application and we believe that a need still exists in the wider scientific community for the easy creation and upload of identifying metadata, so a survey was created to demonstrate this.

To provide more significant validation and show the broader applicability of this software to the national scientific community a survey was conceived and solicited to Environmental Scientists. The survey was comprised of several questions that sought to evaluate how findable, accessible, interoperable, and reusable respondents datasets were. Most questions were bound by a Likert scale of 1 to 5.

The pool of potential respondents came from two working groups called Clusters – for the collaborative organization Environmental Science Information Partners (ESIP). Specifically, members of the Enviroensing Cluster and the Documentation Cluster were emailed with the survey. After hosting the survey for one week 12 responses were acquired.

Using the responses collected from these domain experts we made some preliminary conclusions about how data scientists are managing their metadata. Primarily, we observed that there still exists a strong need for software like the Quality Assurance app to automate and speedup up metadata documentation processes. Secondly, we observed that although metadata was being digitized, it was not being digitized effectively.

Concerning the first conclusion, a few questions demonstrated the current gap

in technology that limits effective metadata collection. When asked how managers document their metadata, the majority of respondents replied with either Field Notebook/ Pen and Paper and Comments on Data Logger Scripts. When we juxtapose these responses against a later question that asks people to estimate the length of time it takes to get metadata into a machine-readable format, a picture emerges showing that scientists are collecting and digitizing this data but not in the most effective way. Upwards of six respondents indicated that it takes them hours and up to weeks to digitize their metadata.

When scientists are extensively using program comments and field journals to collect their data, they are not leveraging the advantages of mobile technology has to offer in the internet age. By comparison, the Quality Assurance app enables the fast and easy input of formatted data which is digital from the start and immediately uploaded to a central database. This application could save many data scientists and technicians hundreds of hours over the course of a year which are now wasted on basic data input.

With this survey and the extensive background works that clearly demonstrate a need to this sort of automated workflow, we have demonstrated that a strong need still exists for software like the QA app presented in this thesis.

7.3 QC Case Study & Discussion

As a part of a case study performed to showcase the proof of concept NRAQC prototype, seven datastreams logging air temperature data were chosen from dozens present within the Rockland Summit site of the Walker Basin Hydroclimate network. It must be noted that the NRAQC prototype can in fact operate on any of the hundreds of datastreams housed inside the greater NRDC system, but these seven were chosen to shorten the runtimes of batch processing required for this case study.

For each of the seven chosen datastreams, 141,951 measurements were cycled through the autonomous tests conducted by NRAQC, 5 times with a manual clearing of the database in between iterations. 993,657 float measurements were batch pro-

cessed in one iteration, with a total of 4,968,285 processed across all five iterations. As each measurement was run through three tests, the total number of tests run per iteration was 2,980,971 with a total of 14,904,855 across all iterations. The average runtime per iteration was 134.24 seconds for this batch processing.

Although not nearly as efficient as possible, the running and re-running of these tests demonstrates the consistency of this system. By running these same measurements through bounds checking tests, repeat value tests, and missing value tests the same points were appropriately flagged each time. In each iteration of testing the backlogged data points, queried from and reinserted directly into the NRDC database, 1946 repeat value flags were logged, 365 out-of-bounds flags were logged, and 131 missing value flags were logged. The QC flags were placed on each data value indicating the test which flagged it. For missing values, empty measurements were created and inserted into the primary data table providing well defined spaces for later data imputation.

These tests specified above are described in Section 2.5.2 and were drawn from the work of Campbell, *et al.* [21]. The parameters of each test were defined through the user interface and stored in an XML configuration file detailed in Chapter 5 and were adjustable without modifying any code in the classes or microservice which performed the tests.

7.3.1 Discussion

In many environmental sensing groups, the notion of Quality Control is not unfamiliar and the practice of using software to handle data sanitation is very common. However, groups tend to use third-party software such as GCE Toolbox alongside the MATLAB Computing Environment instead of creating their own. These software solutions offer a wide array of tools to aid scientists with all sorts of management and control over their datastreams. Although these external software solutions provide significant functionality to suit the purposes of these environmental groups, they come with their own set of problems. Software such as GCE Toolbox tends to be rather rigid

regarding usability. In order to use GCE Toolbox to its maximum potential, the user would be required to have a significant amount of technical knowledge. Additionally, software such as GCE Toolbox is intended to work on a very specialized computing environment, and therefore resists any form of system replication across networked workstations. Finally, most software packages like GCE Toolbox require a sizable fee in order to utilize their services. From a system-level view, this would create far too large of a barrier when it comes to accessibility, procurement, and future maintenance of the software.

The development of a system-level quality control web application addresses each of these problems. By shifting towards the development of a web application, this allows the members of the environmental group to access the services of this system from anywhere with an internet connection. This widens accessibility to this system and it eliminates the need for installation of external software packages on individual terminals.

We developed this software to be completely open sourced and free to the public. By having it open sourced, this handles the procurement problem as any environmental group may simply download the project through a public repository. Additionally, the NRAQC system focuses primarily on the non-technical usability of a quality control system. It allows users to adjust the configuration on virtually all aspects of their datastream through the dashboard without significant need for technical intervention. This intuitive design addresses the problem of future maintenance, as any scientist, technician, or data steward may operate this system without a technical background in computer science or software engineering.

As the fields of sciences, especially environmental sciences, continue to grow, the need for more complex software solutions will increase. The software that engineers build must be able to adapt for larger scales and address the needs of scientists without burdening them with impractical constraints. The NRAQC system provides the means to address the needs of growing environmental sensing projects with its reconfiguration of traditional data management practices.

This software’s primary contribution and innovation comes from its principal design characteristic to be deployed as a core step in a deliberate automated data management workflow. Many common desktop QC solutions, like the GCE Toolbox and Excel, imply a post-ingestion quality control process on data. That is, quality control handled as an afterthought; a step which necessarily precedes analytics to ensure quality results. By decoupling the quality control process from the data collection/ingestion process, crucial quality control metadata is often maintained on local machines or left to languish on closed networks. Data sources, even those from prominent scientific institutions like USGS or CUASHI, will provide data sets without any granular metadata informing data users about what quality control processes each measurement has been verified against [99, 104].

By injecting the Quality Control process automatically into a data-management workflow, metadata is tied fundamentally to its associated data, because they are logged nearly at the same time. The tight coupling of metadata and data forces new projects to accommodate granular metadata in their designs because it’s being collected in volumes and velocities rivaling that of the data collected. This, in turn, leads to a greater accessibility of quality control metadata because it’s intrinsically tied to the data being accessed and used.

This structure aids domain scientists, data users, and data managers in multiple ways. It gives data managers a clear workflow for managing and distributing quality control metadata. It makes it much easier for data users to get a transparent, quality data product with a easily traced provenance showing its pedigree and reassuring them of the validity of their calculations. Finally, domain scientists – who often fund and benefit from from large scale, autonomous data collection projects – are given a infrastructure which enables them to provide clear, understandable data to their colleagues. This, in turn, promotes better reproducibility and independent validation of published research.

7.4 Validation of the Keystone Model

The Keystone Model which was proposed in Chapter 3, suggests that an integrated approach to data quality management would be preferable to current *ad hoc* solutions undertaken by many scientists using various different software solutions. The three software solutions constructed to demonstrate use of this model include several key interactions which also validate its effectiveness.

7.4.1 QA and QC

By having a Quality Assurance application which allows for the immediate input of machine readable-metadata, the ways in which other applications can benefit from this are numerous. Specifically, the NRAQC application benefited from the centralized metadata uploaded by the Quality Assurance application in multiple ways.

First, the metadata uploaded to the NRDC database provided a natural means to organize data streams for configuration. By loading our metadata directly into an organizational hierarchy starting with a Site Network and ending with a single Deployment, users were given a clear and effective means to navigate to a specific data stream they wanted to test. Furthermore, by having this integrated connection, changes made by the Quality Assurance app are immediately reflected in the NRAQC application. This means that seconds after deploying a new instrument and setting up a new stream of measurements, Quality Control processes can be affixed to them with very minimal configuration.

Although novel, the hierarchical navigation also demonstrates another benefit of this shared contextual metadata with the lists to the right of our navigation options. These lists hold all the data managed by the QA application in a read only format, and can help guide data managers towards streams and deployments which have undergone insufficient QA or perhaps a lot of QA, in the form of service. This data can provide users about problem sensors and deployments which require a greater degree of quality control and can be visualized in many ways. The current implementation

of this interaction is not especially practical but it does demonstrate the integration between both applications.

7.4.2 QC and Data Repair

Although no formal integration was done connecting the two programs created to demonstrate QC and Data Repair functionalities, NRAQC and iRSFKM, respectively, the benefit of such a connection is obvious. One of the key tests any quality control software checks for is missing values. The identification of these gaps are crucial to begin the process of repairing the data so a more complete data set can be used for modeling.

Generally this process of data repair can be time consuming and requires significant technical knowledge to implement. If, however this repair happens, with a verifiably accurate algorithm, immediately or shortly after gaps are discovered, while leaving the original data source in check, this obviously reduces the time taken by scientists between data collection and analysis. Of course, since it does not modify the raw data, scientists are free to do their own imputation however for many, this can provide a quick means to do preliminary analysis on a complete data product.

8 Conclusions and Future Work

8.1 Conclusions

In this thesis, we took a deep dive into the technical and domain specific background required to understand its topic, content and focus. As part of this background review of the literature, we highlighted multiple problems facing data management in the earth and environmental sciences. Most notably, problems involving contemporary data Quality Assurance and Quality Control practices. Additionally, we identified key problems with a lack of accurate and fast imputation methods which are configured to work with time-series environmental science data.

To address these issues and promote an integrated, synergistic approach to data management we proposed the Keystone Model of data management. To demonstrate the promise of this model and provide individual solutions to the above identified problems we developed three programs: a mobile Quality Assurance application, an Autonomous Quality Control Application and an Improved Robust and Sparse Fuzzy K-Means algorithm for data imputation.

For our Quality Assurance and Quality Control applications we detailed narratives of work spent on specification, design, and implementation. For the fuzzy K-means imputation algorithm we detailed the iterative process of developing an algorithm from the source of an academic paper and adapting it to run on 8 GPUS.

Individually, we validated these solutions via various methods. First, we validated the Quality Assurance app with the use of a survey that explored the need for such an application in the earth and environmental sciences. It was determined that a strong need still existed for such an application as the one we propose in this thesis.

Furthermore, in a discussion on this topic we asserted that our application fulfills well the requirements expected of such an application by the community.

Next, we assessed the implementation of our Quality Control application by executing a case study where numerous data streams were checked using the autonomous testing modules quintessential to this software. Over the course of five iterations, 4,968,285 measurements were consistently checked against user defined quality metrics. Each time, 1,946 repeat values, 365 out-of-range values, and 131 missing values were identified. This case study proved the consistent execution of the system in fulfilling the purposes and needs outlined by software specifications. Furthermore we outlined the documented need for software of this variety in the earth sciences.

We demonstrated the successful implementation of our iRSFKM algorithm with a comparative analysis of runtimes, throughput, efficiency and speedup between a single CPU implementation, one, two, four, and eight GPU implementations. This experiment — and subsequent comparative analysis — proved the effectiveness of this parallelization by showing that a multi-GPU implementation is significantly faster than sequential implementations with 185 times speedup over eight GPUs. Experiments also indicated greater than 300 times increase in throughput with eight GPUs and 95% efficiency when using two GPUs compared to one.

Finally, the Keystone Model proposed in this thesis was validated with a discussion of the symbiotic relationships our prototype applications share. Additionally, it was argued with examples, that this model predicts potential modifications and improvements to existing data management workflows.

8.2 Future Work

Being a very large project comprised of multiple sub-applications, Keystone introduces significant room for expansion and future work.

8.2.1 Additional Data Imputation Algorithms

In order to prove the value of the Keystone Model *vis-à-vis* the interactions between quality control and data repair, work could be done integrating existing iRSFKM code into the quality control pipeline as an optional last step. Additionally, many other existing data imputation algorithms can be integrated into this pipeline in the place of iRSFKM and compared to evaluate their relative effectiveness. The integration of these works can provide significant value to environmental science researchers who require a rapidly prepared data product to reduce they spend on pre-processing and increase the time spent doing science.

8.2.2 Ontology-Based Microservice & Database Generation

Since both the backend and front end of the NRDC Quality Assurance application has a very constrained structure based around the hierarchy of metadata being stored in our database, there exists significant potential to generate both a backend and fronted based only on a specification of names, fields and relations between classes of metadata. All of this information can be encoded into a single ontology [44], and used a source file to generate custom QA and QC applications which work for specific research projects. If a researcher needs to keep track of their data in a hierarchical fashion — similar to how we manage metadata at the NRDC — software for the end-to-end management of metadata can be generated for easy deployment.

8.2.3 Modifications to the NRDC Schema

Even though the NRAQC software platform is capable of generating very granular information associated with flags, the current NRDC database is not capable of storing it. Information like the date and time of testing, the tests undertaken, and user annotations are currently unsupported by the NRDC database. These columns will need to be added to our extant NRDC metadata tables.

Furthermore, as more than one flag can be applied to one measurement, the relations between flag and measurements tables will need to be modified. Currently,

there is a many-to-one relationship between the measurements and flag tables. This will need to be modified to a logical many-to-many relationship between tables. Practically, this means a dedicated join table will be required to avoid troubles with inner joins and redundant data.

8.2.4 Source Agnostic Data Streaming with Kafka

Congruent to the above detailed future work there exists an opportunity to fundamentally modify the NRDC's current data ingestion workflows to be more sympathetic to integrated, real-time data management processes. Since the inception of the NRDC nearly 10 years ago, there have been significant developments in data streaming technology. Most notably are the introduction of cloud-based streaming management platforms like Kafka and Nifi.

Kafka, a software developed by Apache, introduces a streaming data paradigm based on a market abstraction of “producers” and “consumers” [4]. With this platform, producers put data into a stream with labels that particular consumers look for. For example, in the case of the NRDC, there could be a single producer that ingests data from a SensorML file and puts it into the Kafka data stream labeled as “Raw Data.” This data could be simultaneously consumed by a service which stores it in a database, a service which uploads it to chords for real time visualization and a service which performs real-time quality control checks on the incoming data. After checking this raw data, the quality control service could then become a producer itself and pass quality controlled data back into the system to be used for rapid data imputation or analytics. By leveraging this software platform, the Keystone Model of data management can flow much more effectively, from end-to-end.

Bibliography

- [1] Kazunori Akiyama, Antxon Alberdi, Walter Alef, Keiichi Asada, Rebecca Azu-
lay, Anne-Kathrin Baczko, David Ball, Mislav Baloković, John Barrett, Dan
Bintley, Lindy Blackburn, Wilfred Boland, Katherine L. Bouman, Geoffrey
C. Bower, Michael Bremer, Christiaan D. Brinkerink, Roger Brissenden, Silke
Britzen, Avery E. Broderick, Dominique Broguiere, Thomas Bronzwaer, Do-
Young Byun, John E. Carlstrom, Andrew Chael, Chi-kwan Chan, Shami Chat-
terjee, Koushik Chatterjee, Ming-Tang Chen, Yongjun Chen, Ilje Cho, Pierre
Christian, John E. Conway, James M. Cordes, Geoffrey B. Crew, Yuzhu Cui,
Jordy Davelaar, Mariafelicia De Laurentis, Roger Deane, Jessica Dempsey,
Gregory Desvignes, Jason Dexter, Sheperd S. Doleman, Ralph P. Eatough,
Heino Falcke, Vincent L. Fish, Ed Fomalont, Raquel Fraga-Encinas, William
T. Freeman, Per Friberg, Christian M. Fromm, Jos L. Gómez, Peter Galison,
Charles F. Gammie, Roberto García, Olivier Gentaz, Boris Georgiev, Ciriaco
Goddi, Roman Gold, Minfeng Gu, Mark Gurwell, Kazuhiro Hada, Michael
H. Hecht, Ronald Hesper, Luis C. Ho, Paul Ho, Mareki Honma, Chih-Wei
L. Huang, Lei Huang, David H. Hughes, Shiro Ikeda, Makoto Inoue, Sara
Issaoun, David J. James, Buell T. Jannuzi, Michael Janssen, Britton Jeter,
Wu Jiang, Michael D. Johnson, Svetlana Jorstad, Taehyun Jung, Mansour
Karami, Ramesh Karuppusamy, Tomohisa Kawashima, Garrett K. Keating,
Mark Kettenis, Jae-Young Kim, Junhan Kim, Jongsoo Kim, Motoki Kino,
Jun Yi Koay, Patrick M. Koch, Shoko Koyama, Michael Kramer, Carsten
Kramer, Thomas P. Krichbaum, Cheng-Yu Kuo, Tod R. Lauer, Sang-Sung
Lee, Yan-Rong Li, Zhiyuan Li, Michael Lindqvist, Kuo Liu, Elisabetta Liuzzo,
Wen-Ping Lo, Andrei P. Lobanov, Laurent Loinard, Colin Lonsdale, Ru-Sen
Lu, Nicholas R. MacDonald, Jirong Mao, Sera Markoff, Daniel P. Marrone,
Alan P. Marscher, Ivn Martí-Vidal, Satoki Matsushita, Lynn D. Matthews, Lia
Medeiros, Karl M. Menten, Yosuke Mizuno, Izumi Mizuno, James M. Moran,
Kotaro Moriyama, Monika Moscibrodzka, Cornelia Müller, Hiroshi Nagai, Neil
M. Nagar, Masanori Nakamura, Ramesh Narayan, Gopal Narayanan, Iniyan
Natarajan, Roberto Neri, Chunchong Ni, Aristeidis Noutsos, Hiroki Okino,
Hector Olivares, Gisela N. Ortiz-León, Tomoaki Oyama, Feryal Özel, Daniel
C. M. Palumbo, Nimesh Patel, Ue-Li Pen, Dominic W. Pesce, Vincent Piétu,
Richard Plambeck, Aleksandar PopStefanija, Oliver Porth, Ben Prather, Jorge
A. Preciado-López, Dimitrios Psaltis, Hung-Yi Pu, Venkatesh Ramakrishnan,

- Ramprasad Rao, Mark G. Rawlings, Alexander W. Raymond, Luciano Rezzolla, Bart Ripperda, Freek Roelofs, Alan Rogers, Eduardo Ros, Mel Rose, Arash Roshanineshat, Helge Rottmann, Alan L. Roy, Chet Ruszczyk, Benjamin R. Ryan, Kazi L. J. Rygl, and et. al. First M87 Event Horizon Telescope Results. I. The Shadow of the Supermassive Black Hole. *The Astrophysical Journal*, 875(1):L1, April 2019. ISSN: 2041-8213. DOI: 10.3847/2041-8213/ab0ec7. URL: <http://stacks.iop.org/2041-8205/875/i=1/a=L1?key=\crossref.8b277503073a0a9e72958f0aba518652>.
- [2] Mahmoud Al-Ayyoub, Ansam M Abu-Dalo, Yaser Jararweh, Moath Jarrah, and Mohammad Al Sa'd. A GPU-based implementations of the fuzzy C-means algorithms for medical image segmentation. *The Journal of Supercomputing*, 71(8):3149–3162, August 2015. ISSN: 1573-0484. DOI: 10.1007/s11227-015-1431-y.
- [3] Grigoris Antoniou and Frank van Harmelen. Web Ontology Language: OWL. In *Handbook on Ontologies*, pages 67–92. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. DOI: 10.1007/978-3-540-24750-0{_}4. URL: http://link.springer.com/10.1007/978-3-540-24750-0_4.
- [4] Apache. Apache Kafka. URL: <https://kafka.apache.org/>. Last Accessed on 04/15/19.
- [5] Arizona Geological Survey. California Active Faults - Data.gov. URL: <https://catalog.data.gov/dataset/california-active-faults>. Last Accessed on 05/04/19.
- [6] M. Aubert, A. Brumm, M. Ramli, T. Sutikna, E. W. Saptomo, B. Hakim, M. J. Morwood, G. D. van den Bergh, L. Kinsley, and A. Dosseto. Pleistocene cave art from Sulawesi, Indonesia. *Nature*, 514(7521):223–227, October 2014. ISSN: 0028-0836. DOI: 10.1038/nature13422. URL: <http://www.nature.com/articles/nature13422>.
- [7] S Azim and S Aggarwal. Hybrid model for data imputation: Using fuzzy c means and multi layer perceptron. In *2014 IEEE International Advance Computing Conference (IACC)*, pages 1281–1285. DOI: 10.1109/IAdCC.2014.6779512.
- [8] Tanvi Banerjee, James M Keller, Marjorie Skubic, and Erik Stone. Day or night activity recognition from video using fuzzy clustering techniques. *IEEE Transactions on Fuzzy Systems*, 22(3):483–493, 2014.
- [9] Derik Barseghian, Ilkay Altintas, Matthew B Jones, Daniel Crawl, Nathan Potter, James Gallagher, Peter Cornillon, Mark Schildhauer, Elizabeth T Borer, and Eric W Seabloom. Workflows and extensions to the kepler scientific workflow system to support environmental sensor data access and analysis. *Ecological Informatics*, 5(1):42–50, 2010.
- [10] Battelle. NSF NEON — Open Data to Understand our Ecosystems, 2019. URL: <https://www.neonscience.org/>. Last Accessed on 05/01/19.

- [11] Mike Beauregard. left wall of the hall of bulls — Life-size replica from Lasceaux, France — Flickr, 2017. URL: <https://www.flickr.com/photos/31856336@N03/35992500056/>. Last Accessed on 05/01/19.
- [12] Sean Bechhofer, Sean Bechhofer, David De Roure, Matthew Gamble, Carole Goble, and Iain Buchan. Research Objects: Towards Exchange and Reuse of Digital Knowledge. *Nature Precedings*, July 2010. ISSN: 1756-0357. DOI: 10.1038/npre.2010.4626.1. URL: <http://precedings.nature.com/doifinder/10.1038/npre.2010.4626.1>.
- [13] Lorenzo Beretta and Alessandro Santaniello. Nearest neighbor imputation algorithms: a critical evaluation. *BMC medical informatics and decision making*, 16(3):74, 2016.
- [14] Franco Biondi, Leia P. Jamieson, Scotty Strachan, and Jason Sibold. Dendroecological testing of the pyroclimatic hypothesis in the central Great Basin, Nevada, USA. *Ecosphere*, 2(1):art5, January 2011. ISSN: 2150-8925. DOI: 10.1890/ES10-00068.1. URL: <http://doi.wiley.com/10.1890/ES10-00068.1>.
- [15] Wade Bishop and Tony H Grubestic. Metadata. In *Geographic Information*, pages 79–103. Springer, 2016.
- [16] Mike Bostock. Data-Driven Documents, 2018. URL: <https://d3js.org/>. Last Accessed on 05/01/19.
- [17] Stephen Boyd and Lieven Vandenbergh. *Convex Optimization*. Cambridge University Press, Cambridge, 7th edition, 2009, pages 1–16. URL: <https://web.stanford.edu/~boyd/cvxbook/>.
- [18] Tim Bray, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, and Francois Yergeau. Extensible markup language (xml) 1.0, 2000.
- [19] Dan Brickley and Libby Miller. Foaf vocabulary specification 0.99 (2014). *Namespace Document*. Available online: <http://xmlns.com/foaf/spec/> (accessed on 23 November 2018), 2015.
- [20] David Brillinger. *Time Series: Data Analysis and Theory - David R. Brillinger - Google Books*. Society for Industrial and Applied Mathematics, San Fransisco, 10th edition, 2001, pages 1–16. ISBN: 978-0-89871-501-9. URL: https://books.google.com/books?hl=en&lr=&id=3DFJfgEW94gC&oi=fnd&pg=PR3&dq=time+series+data&ots=WbE66n91I1&sig=RsSM8dpyD53VfNrs_TgWePlGgBY#v=onepage&q=time%20series%20data&f=false.
- [21] John L Campbell, Lindsey E Rustand, John H Porter, Jeffery R Taylor, Ethan W Dereszynski, James B Shanley, Corinna Gries, Donald L Henshaw, Mary E Martin, and Wade M Sheldon. Quantity is Nothing without Quality. *BioScience*, 63(7):574–585, 2013. DOI: 10.1525/bio.2013.63.7.10.
- [22] National Weather Service US Department of Commerce NOAA. National Forecast Maps. URL: <https://www.weather.gov/forecastmaps>.

- [23] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). Technical report, July 2006. DOI: 10.17487/rfc4627. URL: <https://www.rfc-editor.org/info/rfc4627>. Last Accessed on 05/16/19.
- [24] CUHASI. Find, Analyze and Share Water Data — CUAHSI HydroShare. URL: <https://www.hydroshare.org/>. Last Accessed on 05/01/19.
- [25] Richard Cyganiak, David Wood, Markus Lanthaler, Graham Klyne, Jeremy J Carroll, and Brian McBride. Rdf 1.1 concepts and abstract syntax. *W3C recommendation*, 25(02), 2014.
- [26] Steven Diamond and Stephen Boyd. CVXPY: A Python-Embedded Modeling Language for Convex Optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.
- [27] Docker. Docker. URL: <https://www.docker.com/>. Last Accessed on 05/16/19.
- [28] Docker Docs. Compose file version 3 reference — Docker Documentation. URL: <https://docs.docker.com/compose/compose-file/>. Last Accessed on 04/18/19.
- [29] DOI.org. Digital Object Identifier System Handbook. Technical report, DOI, 2012. DOI: 10.1000/182. URL: <https://www.doi.org/hb.html>. Last Accessed on 05/16/19.
- [30] A Domahidi, E Chu, and S Boyd. ECOS: An SOCP solver for embedded systems. In *European Control Conference (ECC)*, pages 3071–3076.
- [31] Dublin Core. DCMI: DCMI Metadata Terms. DOI: <http://dublincore.org/specifications/dublin-core/dcmi-terms/2012-06-14/>. URL: <http://dublincore.org/specifications/dublin-core/dcmi-terms/2012-06-14/?v=terms#abstract>. Last Accessed on 04/12/19.
- [32] J C Dunn. A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters. *Journal of Cybernetics*, 3(3):32–57, 1973. DOI: 10.1080/01969727308546046.
- [33] Earth Cube. Abstract — EarthCube, 2018. URL: <https://www.earthcube.org/>. Last Accessed on 04/12/19.
- [34] Envirosensing Cluster. Sensor Data Management Middleware - Federation of Earth Science Information Partners, 2019. URL: http://wiki.esipfed.org/index.php/Sensor_Data_Management_Middleware. Last Accessed on 04/12/19.
- [35] ESIP. Federation of Earth Science Information Partners, 2018. URL: http://wiki.esipfed.org/index.php/Main_Page. Last Accessed on 04/12/19.
- [36] ESIP. Mission, Vision and Values — ESIP, 2019. URL: <https://www.esipfed.org/about/mission>. Last Accessed on 04/12/19.
- [37] ESIP Envirosensing Cluster. EnviroSensing Cluster - Federation of Earth Science Information Partners, 2019. URL: http://wiki.esipfed.org/index.php/EnviroSensing_Cluster. Last Accessed on 04/12/19.

- [38] ESRI. ArcGIS, October 2017. URL: <http://resources.arcgis.com/en/communities/data-reviewer/01rp0000006000000.htm>. Last Accessed on 04/12/19.
- [39] ESRI. Data quality and validation ArcGIS Pro — ArcGIS Desktop. URL: <https://pro.arcgis.com/en/pro-app/help/data/validating-data/data-quality-and-validation-in-arcgis-professional.htm>. Last Accessed on 04/12/19.
- [40] Oskar M Essenwanger. Analytical Procedures for the Quality Control of Meteorological Data. *Meteorological Observations and Instrumentation*:141–147, 1970. DOI: 10.1007/978-1-935704-35-5_19.
- [41] R Fielding, J Gettys, J Mogul, H Frystyk, L Masinter, P Leach, and T Berners-Lee. HTTP/1.1: Request, 1999. URL: <https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html>. Last Accessed on 05/16/19.
- [42] Thomas Foken, Mathias Göckede, Matthias Mauder, Larry Mahrt, Brian Amiro, and William Munger. Post-Field Data Quality Control. *Handbook of Micrometeorology Atmospheric and Oceanographic Sciences Library*:181–208, 2004. DOI: 10.1007/1-4020-2265-4_9.
- [43] FORCE 11. About FORCE11 — FORCE11, 2019. URL: <https://www.force11.org/about>. Last Accessed on 04/14/19.
- [44] Sandra Geisler, Christoph Quix, Sven Weber, and Matthias Jarke. Ontology-Based Data Quality Management for Data Streams. *Journal of Data and Information Quality*, 7(4):1–34, October 2016. ISSN: 19361955. DOI: 10.1145/2968332. URL: <http://dl.acm.org/citation.cfm?doid=3006343.2968332>.
- [45] Google. Angular Docs, 2018. URL: <https://angular.io/>. Last Accessed on 04/14/19.
- [46] Sean Gordon and Ted Habermann. The influence of community recommendations on metadata completeness. *Ecological informatics*, 43:38–51, 2018.
- [47] Carl C Gouldman, Kathleen Bailey, and Julianna O Thomas. Manual for Real-Time Oceanographic Data Quality Control Flags. *IOOS*, 2017. URL: https://ioos.noaa.gov/wp-content/uploads/2017/06/QARTOD-Data-Flags-Manual/_Final_version1.1.pdf. Last Accessed on 05/16/19.
- [48] R.V. Guha and Dan Brickley. About schema.org. URL: <https://schema.org/docs/about.html>. Last Accessed on 04/12/19.
- [49] David J Hill and Barbara S Minsker. Anomaly detection in streaming environmental sensor data: A data-driven modeling approach. *Environmental Modelling & Software*, 25(9):1014–1022, September 2010. DOI: 10.1016/j.envsoft.2009.08.010.

- [50] John E. Hobbie, Stephen R. Carpenter, Nancy B. Grimm, James R. Gosz, and Timothy R. Seastedt. The US Long Term Ecological Research Program. *BioScience*, 53(1):21–32, January 2003. ISSN: 0006-3568. DOI: 10.1641/0006-3568(2003)053[0021:tulter]2.0.co;2. URL: <https://academic.oup.com/bioscience/article/53/1/21/227145>.
- [51] Jin Huang, Feiping Nie, and Heng Huang. A New Simplex Sparse Learning Model to Measure Data Similarity for Clustering. In *Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI'15*, pages 3569–3575. AAAI Press. ISBN: 978-1-57735-738-4.
- [52] Ionic. Open Source Framework for Building Amazing Mobile Apps, 2019. URL: <https://ionicframework.com/framework>. Last Accessed on 04/12/19.
- [53] Matthew B. Jones, Mark P. Schildhauer, O.J. Reichman, and Shawn Bowers. The New Bioinformatics: Integrating Ecological Data from the Gene to the Biosphere. *Annual Review of Ecology, Evolution, and Systematics*, 37(1):519–544, December 2006. ISSN: 1543-592X. DOI: 10.1146/annurev.ecolsys.37.091305.110031. URL: <http://www.annualreviews.org/doi/10.1146/annurev.ecolsys.37.091305.110031>.
- [54] Hobart M. King. What is Earth Science? — Geology.com, 2019. URL: <https://geology.com/articles/what-is-earth-science.shtml>. Last Accessed on 05/04/19.
- [55] John Kratz and Carly Strasser. Data publication consensus and controversies. *F1000 Research*, 3, 2014. Last Accessed on 05/16/19.
- [56] Vinh Dac Le, Melanie M Neff, Richard V Stewart, Richard Kelley, Eric Fritzinger, Sergiu M Dascalu, and Frederick C Harris. Microservice-based architecture for the NRDC. In *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, pages 1659–1664, July 2015. DOI: 10.1109/INDIN.2015.7281983.
- [57] Dan Li, Jitender Deogun, William Spaulding, and Bill Shuart. Towards Missing Data Imputation: A Study of Fuzzy K-means Clustering Method. In Shusaku Tsumoto, Roman Słowiński, Jan Komorowski, and Jerzy W Grzymała-Busse, editors, *Rough Sets and Current Trends in Computing*, pages 573–579, Berlin, Heidelberg. Springer Berlin Heidelberg. ISBN: 978-3-540-25929-9.
- [58] Zaifei Liao, Xinjie Lu, Tian Yang, and Hongan Wang. Missing data imputation: a fuzzy K-means clustering algorithm over sliding window. In *Fuzzy Systems and Knowledge Discovery, 2009. FSKD'09. Sixth International Conference on*, volume 3, pages 133–137. IEEE.
- [59] Georgia Coastal Ecosystems LTER. GCE Data Toolbox for MATLAB, 2017. URL: http://gce-lter.marsci.uga.edu/public/im/tools/data/_toolbox.htm.

- [60] Ilias G. Maglogiannis. *Emerging artificial intelligence applications in computer engineering : real word AI systems with applications in eHealth, HCI, information retrieval and pervasive technologies*. IOS Press, 2007, page 407. ISBN: 9781586037802. URL: https://books.google.com/books?hl=en&lr=&id=vLiTXDhr_sYC&oi=fnd&pg=PA3&dq=supervised+machine+learning+algorithms&ots=CYPxtv4Bni&sig=C7P6ufBo7Ps_cJLEIdrQ17BqhwQ#v=onepage&q=supervised%20machine%20learning%20algorithms&f=false.
- [61] Bernard Marr. How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read, 2018. URL: <https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#3fba15cf60ba>. Last Accessed on 05/16/19.
- [62] Larry Masinter, Tim Berners-Lee, and Roy T. Fielding. Uniform Resource Identifier (URI): Generic Syntax. URL: <https://tools.ietf.org/html/rfc3986>. Last Accessed on 05/16/19.
- [63] MathWorks. MATLAB - MathWorks - MATLAB & Simulink, 2019. URL: <https://www.mathworks.com/products/matlab.html>. Last Accessed on 05/02/19.
- [64] Jacob Mattingley and Stephen Boyd. CVXGEN: a code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, March 2012. ISSN: 1573-2924. DOI: 10.1007/s11081-011-9176-9.
- [65] Michael J. McMahon, Frederick C. Harris, Sergiu M. Dascalu, and Scotty Strachan. S.E.N.S.O.R. applying modern software and data management practices to climate research. In *Proceedings of the 2011 Workshop on Senesor Network Applications November 16-18, 2011, Honolulu, HI*. 2011.
- [66] William K Michener. Meta-information concepts for ecological data management. *Ecological Informatics*, 1(1):3–7, January 2006. DOI: 10.1016/j.ecoinf.2005.08.004.
- [67] William K. Michener. Quality Assurance and Quality Control (QA/QC). In *Ecological Informatics*, pages 55–70. Springer International Publishing, Cham, 2018. DOI: 10.1007/978-3-319-59928-1_4. URL: http://link.springer.com/10.1007/978-3-319-59928-1_4.
- [68] Rakhi Motwani, Mukesh Motwani, Frederick Harris Jr., and Sergiu Dascalu. Towards a scalable and interoperable global environmental sensor network using service oriented architecture. In *2010 Sixth International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pages 151–156, 2010. DOI: 10.1109/ISSNIP.2010.5706788.
- [69] Hannah Munoz, Connor Scully-Allison, Vinh Le, Scotty Strachan, Fredrick C. Harris, and Sergiu Dascalu. A mobile quality assurance application for the NRDC. In *26th International Conference on Software Engineering and Data Engineering, SEDE 2017*, 2017. ISBN: 9781943436095.

- [70] NASA. Earth Observation Data — Earthdata. URL: <https://earthdata.nasa.gov/earth-observation-data>. Last Accessed on 05/02/19.
- [71] UCAR NCAR. Welcome - NCAR DASH Search. URL: <https://data.ucar.edu/>. Last Accessed on 05/02/19.
- [72] Nevada Climate Change Portal. URL: <http://sensor.nevada.edu/NCCP/>. Last Accessed on 05/02/19.
- [73] NGDATA. What is Data Management? URL: <https://www.ngdata.com/what-is-data-management/>. Last Accessed on 05/02/19.
- [74] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with CUDA. *Queue*, 6(2):40, March 2008. ISSN: 15427730. DOI: 10.1145/1365490.1365500. URL: <http://portal.acm.org/citation.cfm?doid=1365490.1365500>.
- [75] NOAA. National Oceanic and Atmospheric Administration. URL: <http://www.noaa.gov/>. Last Accessed on 05/02/19.
- [76] Marco S. Nobile, Paolo Cazzaniga, Daniela Besozzi, Dario Pescini, and Giancarlo Mauri. cuTauLeaping: A GPU-Powered Tau-Leaping Stochastic Simulator for Massive Parallel Analyses of Biological Systems. *PLoS ONE*, 9(3):e91963, March 2014. Jean Peccoud, editor. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0091963. URL: <https://dx.plos.org/10.1371/journal.pone.0091963>.
- [77] NRDC. Nevada Research Data Center, 2018. URL: <http://sensor.nevada.edu/NRDC/>. Last Accessed on 05/02/19.
- [78] PyCUDA. Welcome to PyCUDAs documentation! PyCUDA 2018.1.1 documentation, 2019. URL: <https://document.tician.de/pycuda/>. Last Accessed on 05/04/19.
- [79] Python. About Python — Python.org, 2019. URL: <https://www.python.org/about/>. Last Accessed on 05/04/19.
- [80] Muhammad Raza. Containers vs Virtual Machines: Whats The Difference? BMC Blogs, 2018. URL: <https://www.bmc.com/blogs/containers-vs-virtual-machines/>. Last Accessed on 05/01/19.
- [81] Dominique G. Roche, Loeske E. B. Kruuk, Robert Lanfear, and Sandra A. Binning. Public Data Archiving in Ecology and Evolution: How Well Are We Doing? *PLOS Biology*, 13(11):e1002295, November 2015. ISSN: 1545-7885. DOI: 10.1371/journal.pbio.1002295. URL: <https://dx.plos.org/10.1371/journal.pbio.1002295>.
- [82] Armin Ronacher. Flask: web development, one drop at a time, 2018. URL: <http://flask.pocoo.org/>. Last Accessed on 05/04/19.
- [83] Peter Schmitt, Jonas Mandel, and Mickael Guedj. A Comparison of Six Methods for Missing Data Imputation. *Journal of Biometrics & Biostatistics*, 06, May 2015. DOI: 10.4172/2155-6180.1000224.

- [84] Campbell Scientific. LoggerNET, December 2017. URL: <https://www.campbellsci.com/loggernet>. Last Accessed on 02/11/19.
- [85] SciPy Developers. Scientific Computing Tools for Python SciPy.org, 2019. URL: <https://www.scipy.org/about.html>. Last Accessed on 05/04/19.
- [86] C. Scully-Allison, V. Le, E. Fritzinger, S. Strachan, F.C. Harris, and S.M. Dascalu. Near Real-time Autonomous Quality Control for Streaming Environmental Sensor Data. In *Procedia Computer Science*, volume 126, 2018. DOI: 10.1016/j.procs.2018.08.139.
- [87] Connor Scully-Allison, Vinh Le, Scotty Strachan, Frederick C Harris Jr., and Sergiu Dascalu. A Mobile Quality Assurance Application for the NRDC. *Proceedings of the ISCA 26th International Conference on Software Engineering and Data Engineering (SEDE 2017)*:185–192, October 2017.
- [88] Connor Scully-Allison, Hannah Muñoz, Vinh Le, and Scotty Strachan. Advancing Quality Assurance Through Metadata Management : Design and Development of a Mobile Application for the NRDC. *International Journal of Computers and Their Applications (IJCA)*, 25(1):20–29, 2018.
- [89] Connor Scully-Allison, Rui Wu, Sergiu M Dascalu, Lee Barford, and Frederick C Harris. Data Imputation with an Improved Robust and Sparse Fuzzy K-Means Algorithm. In *Advances in Intelligent Systems and Computing*. Volume 800, Chapter 41, pp 299-306. *Proceedings of the 16th International Conference on Information Technology : New Generations (ITNG 2019)* April 1-3, Las Vegas, NV.
- [90] S A Arul Shalom, Manoranjan Dash, and Minh Tue. Efficient K-Means Clustering Using Accelerated Graphics Processors. In Il-Yeol Song, Johann Eder, and Tho Manh Nguyen, editors, *Data Warehousing and Knowledge Discovery*, pages 166–175, Berlin, Heidelberg. Springer Berlin Heidelberg. ISBN: 978-3-540-85836-2.
- [91] S A Arul Shalom, Manoranjan Dash, and Minh Tue. Graphics Hardware Based Efficient and Scalable Fuzzy C-means Clustering. In *Proceedings of the 7th Australasian Data Mining Conference - Volume 87*, AusDM '08, pages 179–186, Darlinghurst, Australia, Australia. Australian Computer Society, Inc. ISBN: 978-1-920682-68-2.
- [92] Stall Shelley. Enabling fair project overview COPDESS. URL: <https://copdess.org/enabling-fair-data-project/enabling-fair-project-overview/>. Last Accessed on 05/04/19.
- [93] Nate Silver. 2018 House Forecast — FiveThirtyEight. URL: <https://projects.fivethirtyeight.com/2018-midterm-election-forecast/house/>. Last Accessed on 04/30/19.
- [94] Marina Soley-Bori. Dealing with missing data: Key assumptions and methods for applied analysis. Technical report 4, Boston University, School of Public Health, Department of Health Policy and Management, May 2013.

- [95] Devin Soni. Supervised vs. Unsupervised Learning, March 2018. URL: <https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d>. Last Accessed on 04/30/19.
- [96] Scotty Strachan. Walker Basin Hydroclimate. URL: <http://sensor.nevada.edu/WalkerBasinHydro/>. Last Accessed on 04/30/19.
- [97] C. A. Strasser and S. E. Hampton. The fractured lab notebook: undergraduates and ecological data management training in the United States. *Ecosphere*, 3(12):art116, December 2012. ISSN: 2150-8925. DOI: 10.1890/ES12-00139.1. URL: <http://doi.wiley.com/10.1890/ES12-00139.1>.
- [98] Jinjun Tang, Guohui Zhang, Yinhai Wang, Hua Wang, and Fang Liu. A hybrid approach to integrate fuzzy C-means based imputation method with genetic algorithm for missing traffic volume data estimation. *Transportation Research Part C: Emerging Technologies*, 51:29–40, February 2015. ISSN: 0968-090X. DOI: <https://doi.org/10.1016/j.trc.2014.11.003>.
- [99] David G Tarboton, Jeffery S Horsburgh, and David R Maidment. CUAHSI Community Observations Data Model (ODM) Version 1.1 Design Specifications. Technical report, 2008. URL: https://www.cuahsi.org/uploads/pages/img/ODM1.1DesignSpecifications_.pdf.
- [100] Carol Tenopir, Suzie Allard, Kimberly Douglass, Arsev Umur Aydinoglu, Lei Wu, Eleanor Read, Maribeth Manoff, and Mike Frame. Data sharing by scientists: practices and perceptions. *PloS one*, 6(6):e21101, 2011.
- [101] The R Foundation. R: What is R?, 2019. URL: <https://www.r-project.org/about.html>. Last Accessed on 04/30/19.
- [102] Scott Tyler. Hydrologic Impacts of Forest Thinning — Scott Tyler, Hydrologist. URL: <https://scotttylerhydro.com/current-projects/hydrologic-impacts-of-forest-thinning/>. Last Accessed on 04/30/19.
- [103] TypeScript. TypeScript - JavaScript that scales. 2019. URL: <https://www.typescriptlang.org/>. Last Accessed on 04/30/19.
- [104] USGS. USGS, 2018. URL: <https://waterdata.usgs.gov/nwis>. Last Accessed on 04/30/19.
- [105] USGS WaterWatch – Streamflow conditions. URL: <https://waterwatch.usgs.gov/>. Last Accessed on 04/30/19.
- [106] Faramarz Valafar. Pattern recognition techniques in microarray data analysis: A Survey. *Annals of the New York Academy of Sciences*, 980(1):41–64, 2002.
- [107] Michael C Whitlock. Data archiving in ecology and evolution: best practices. *Trends in Ecology & Evolution*, 26(2):61–65, 2011.

- [108] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Merc Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J.G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A.C t Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3:160018, March 2016. ISSN: 2052-4463. DOI: 10.1038/sdata.2016.18. URL: <http://www.nature.com/articles/sdata201618>.
- [109] Matthew Williams, Dan Cornford, Lucy Bastin, Richard Jones, and Stephen Parker. Automatic processing, quality assurance and serving of real-time weather data. *Computers & Geosciences*, 37(3):353–362, 2011.
- [110] Jinglin Xu, Junwei Han, Kai Xiong, and Feiping Nie. Robust and Sparse Fuzzy K-means Clustering. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI'16*, pages 2224–2230. AAAI Press. ISBN: 978-1-57735-770-4.