Spring 1-1-2012

Analysis and Implementation of Software Tools to Support Research in Crisis Informatics

Casey McTaggart

University of Colorado at Boulder, casey.mctaggart@colorado.edu

Follow this and additional works at: https://scholar.colorado.edu/csci_gradetds

Part of the Computer Sciences Commons

Recommended Citation

https://scholar.colorado.edu/csci_gradetds/40

This Thesis is brought to you for free and open access by Computer Science at CU Scholar. It has been accepted for inclusion in Computer Science Graduate Theses & Dissertations by an authorized administrator of CU Scholar. For more information, please contact cuscholaradmin@colorado.edu.
Analysis and Implementation of Software Tools to Support Research in Crisis Informatics

Casey McTaggart

B.S., Princeton University, 2001

M.S., University of California, Berkeley, 2009

M.S., University of Colorado, Boulder, 2012

A thesis submitted to the
Faculty of the Graduate School of the
University of Colorado, Boulder in partial fulfillment
of the requirement for the degree of
Master of Science

Department of Computer Science

2012
This thesis entitled:
Analysis and Implementation of Software Tools to Support Research in Crisis Informatics
Written by Casey McTaggart
has been approved for the Department of Computer Science

Kenneth Anderson

Leysia Palen

James Martin

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline
McTaggart, Casey (M.S., Computer Science)

Analysis and Implementation of Software Tools to Support Research in Crisis Informatics

Thesis directed by Associate Professor Kenneth Anderson and Professor Leysia Palen

The field of crisis informatics is expanding, in large part because emergency response has become increasingly focused on and affected by social media. Correspondingly, social media datasets collected from disaster events continue to become larger and more intractable. Emergency response events can generate data sets with millions of entries; sifting through the data by hand is no longer feasible. Instead, crisis informatics researchers require software that allows them to perform statistical analysis and filtering operations to reduce data sets to more manageable sizes. This thesis presents an analysis of what software is best for crisis informatics analysts, beginning with the initial process of determining the necessary tools and services, the subsequent implementation, and finally a usability study.
Acknowledgements

I would like to thank my advisers Leysia Palen and Ken Anderson for their continued guidance and direction.

I would also like to thank Aaron Schram for everything.

Thank you to all of the analysts that helped me in this work: Leysia Palen, Joanne White, Sarah Vieweg, Catharine Starbird, Sudha Verma, Joshua Barron and Lise St. Denis.

Thank you to Project EPIC for providing funding for me to continue this research.
### Table of Contents

1. Introduction .................................................................................................................. 1  
2. Motivation and goals...................................................................................................... 2  
3. Current research practice.............................................................................................. 3  
   3.1 Analyst reported research practice ................................................................. 3  
   3.2 Research practice findings ............................................................................. 6  
   3.3 Published research practices and related work ............................................. 7  
4. Investigative user research ......................................................................................... 10  
   4.1 Interview questions, answers, and commonalities ........................................... 11  
      4.1.1 General research interest ........................................................................ 11  
      4.1.1.1 Common findings ............................................................................ 11  
      4.1.1.2 Individual responses ...................................................................... 11  
      4.1.2 Dataset information ............................................................................. 12  
      4.1.2.1 Common findings ............................................................................ 12  
      4.1.2.2 Individual responses ...................................................................... 13  
      4.1.3 Classifying a good dataset ...................................................................... 14  
      4.1.3.1 Common findings ............................................................................ 14  
      4.1.3.2 Individual responses ...................................................................... 14  
      4.1.4 Dataset usability ..................................................................................... 15  
      4.1.5 Blackout periods ..................................................................................... 16  
      4.1.5.1 Common findings ............................................................................ 16  
      4.1.5.2 Individual responses ...................................................................... 16  
      4.1.6 Requirements elicitation ......................................................................... 16  
      4.1.6.1 Common findings ............................................................................ 17  
      4.1.6.2 Individual responses ...................................................................... 18  
      4.1.7 Prior workflow ......................................................................................... 19  
      4.1.8 Ideal software implementation ............................................................... 19  
      4.1.8.1 Common findings ............................................................................ 20  
      4.1.8.2 Individual responses ...................................................................... 20  
   4.2 Mockup response ................................................................................................. 21  
5. Evolution of requirements ......................................................................................... 22
### List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Tweet attributes stored in MySQL</td>
<td>40</td>
</tr>
<tr>
<td>6.2</td>
<td>Mention attributes stored in MySQL</td>
<td>40</td>
</tr>
<tr>
<td>6.3</td>
<td>TwitterQuery attributes stored in MySQL</td>
<td>40</td>
</tr>
<tr>
<td>7.1</td>
<td>Difference between Estimate Reduction and Submit Filter Operations</td>
<td>55</td>
</tr>
</tbody>
</table>
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>Heavytail sampling pseudo-code</td>
<td>27</td>
</tr>
<tr>
<td>6.1</td>
<td>Relationships between Event, TwitterQueries and Tweets</td>
<td>29</td>
</tr>
<tr>
<td>6.2</td>
<td>Architecture interaction with Lucene</td>
<td>32</td>
</tr>
<tr>
<td>6.3</td>
<td>Screenshot of search on epic-collect version 1</td>
<td>33</td>
</tr>
<tr>
<td>6.4</td>
<td>Home page tree view</td>
<td>34</td>
</tr>
<tr>
<td>6.5</td>
<td>Event editing view for admin users</td>
<td>34</td>
</tr>
<tr>
<td>6.6</td>
<td>Event editing for analyst user</td>
<td>35</td>
</tr>
<tr>
<td>6.7</td>
<td>The analyst application tiered structure</td>
<td>37</td>
</tr>
<tr>
<td>6.8</td>
<td>Architecture interaction between Cassandra collection software and analyst application – collection</td>
<td>37</td>
</tr>
<tr>
<td>6.9</td>
<td>Architecture interaction between Cassandra collection software and analyst application – database access</td>
<td>38</td>
</tr>
<tr>
<td>6.10</td>
<td>Architecture interaction between Cassandra collection software and analyst application – analyst database access</td>
<td>38</td>
</tr>
<tr>
<td>6.11</td>
<td>The new analyst application architecture</td>
<td>39</td>
</tr>
<tr>
<td>6.12</td>
<td>Relationship between TwitterQueries, Tweets and Mentions</td>
<td>41</td>
</tr>
<tr>
<td>6.13</td>
<td>Maintaining filtered datasets</td>
<td>42</td>
</tr>
<tr>
<td>6.14</td>
<td>Descriptive Statistics screenshot</td>
<td>43</td>
</tr>
<tr>
<td>6.15</td>
<td>Filters page screenshot</td>
<td>44</td>
</tr>
<tr>
<td>6.16</td>
<td>Sampling page screenshot</td>
<td>45</td>
</tr>
<tr>
<td>6.17</td>
<td>Graph of volume of tweets over time</td>
<td>47</td>
</tr>
<tr>
<td>6.18</td>
<td>Graph of users and volume</td>
<td>47</td>
</tr>
</tbody>
</table>
1 Introduction

The expanding field of crisis informatics is an area of research and development that “addresses socio-technical concerns in large scale emergency response, [expanding] consideration to include not only official responders... but also members of the public” (Palen, et al 2010). Crisis informatics spans preparation, warning, response and recovery, with a view of “emergency response as a social system where information is disseminated within and between official and public channels and entities” (Palen et al, 2010). Project EPIC has been at the forefront of crisis informatics research; the group is leading the effort to collect data during crisis events, analyze this data, and construct important research questions. As per the mission statement of empowering the public with information in crisis, Project EPIC focuses on public needs, with an eye towards a greater understanding of how and why the public communicates. This thesis work has been defined by the needs and goals of the analysts working for Project EPIC.

A current focus for Project EPIC is on data collected from the microblogging site Twitter (Twitter, 2012). When a major event occurs, the collection software is fed appropriate search terms. These are terms that are prevalent and relevant to the dissemination of information regarding the event. Sometimes these terms change as the event changes, and the analysts are responsible for monitoring and updating search terms as necessary. Ultimately, the hope is that an event database contains as many tweets as possible about the event.

Once a crisis event is determined to be over, a more focused analysis begins. This includes statistical analysis as well as ad hoc methods of filtering and sampling. Working with such large data poses significant challenges in both determining how to frame appropriate research questions as well as how to architect software solutions. The
focus of this thesis is to investigate and implement the tools and services needed to support research in crisis informatics, given the constraints of working with datasets numbering in the millions of tweets.

The work for this thesis proceeded as follows. First, I interviewed analysts to gain an understanding of current work practice. This helped in identifying the problem domain. Then, I conducted an investigative study to elicit software requirements from the analysts to facilitate their research. After this initial investigative effort, I mocked up a UI, reviewed it with the analysts, and solidified the requirements. At this point the software engineering team discussed software architecture changes, which involved splitting the old application into two components: collection and analysis. The analyst application was defined as the component that will be used to perform statistical analysis, filtering and sampling on events once they have finished. After an initial software effort on the analyst application, I conducted a usability study to determine what else was required of the application. Bugs were fixed and features were added as a result of the usability study. Finally, the application was deployed for further usability testing and feedback from the analysts. Ideally, there will continue to be iteration between software engineering and the analysts to converge on the optimal set of tools.

2 Motivation and goals

The goal of this thesis is to provide an initial implementation of tools and services required by analysts for research in crisis informatics. These tools and services are informed by both the analysts’ current research practice and the results of the usability

1 Myself, Kenneth Anderson and Aaron Schram
study; I used both in identifying what will best help them conduct their research going forward.

The investigative study began the process. It was motivated by the outcome of a series of meetings with the analysts that highlighted the difficulties they were encountering in both using the old architecture and in completing the work they wished to do. Prior to the new architecture, every time an analyst wanted data, someone on the software engineering team had to manipulate data in a specific way to be a reasonable size and format. As events continued to unfold, and as it was clear that each analyst had different data requirements, the P.I. of Project EPIC\(^2\) suggested that a study be run on the current application to determine (a) current workflow, (b) usability of the old application, and (c) requirements for a new application that would better facilitate analyst research.

Once this initial study was completed, it was then the task of the Software Engineering team to rearchitect the software to fit the new requirements\(^3\). On rearchitecture completion, the new interface and functionality was shown again to the analysts. They provided feedback on positive and negative components of the architecture, and these comments were incorporated into software iteration where time and resources permitted. I facilitated this process of usability testing coupled with software development, with the goal of building a final product both usable for the analysts and maintainable for the software engineering team.

3 Current research practice

3.1 Analyst reported research practice

\(^2\) Principal Investigator Leysia Palen
\(^3\) The evolution of software requirements is described in detail in Section 5
Currently, each analyst has a different method of working, since each analyst is asking a different research question. Five analysts were consulted in this investigative effort. I determined this was a complete set of interviews, since at that time there were five analysts conducting research in this space. Each analyst was asked about research practice as it exists outside of the analyst application. The analysts are referred to anonymously henceforth as analysts A1 – A5.

Analyst A1 prefers to work directly with the Twitter website. She is continually monitoring the stream for trending hashtags, and when she notes one of interest, she will usually follow it, or scan it for interesting users. She also will execute searches on her specific research topics of interest, and take note of things such as which Twitterers used a certain word, how many times that word came up, and how popular it was. For each event, she depends on members of the software engineering team to do the filtering and sampling of a dataset such that it is loadable in Excel. Once she has acquired a sufficiently filtered dataset of tweets, she continues her exploration in Excel, performing her own methods of searching, and identifying interesting trends depending on what she observes in the dataset.

Analyst A2 reports first researching the actual event, using tools such as FEMA situational reports and journalistic articles on the web. She is looking for information such as where the event started, when it started, and what caused it; and she is hoping to get both an overall picture of the event as well as some insight into research questions. She then looks to the software engineering team to obtain filtered data. The way she wants this filtering to occur varies by event. For example, for the Haiti earthquake event, all tweets that included a predetermined set of terms or strings were
filtered from the original set of 4 million. Then, duplicate tweets and retweets were eliminated. To further whittle down the data set, she deployed a situational awareness classifier on the remaining tweets (Vieweg 2012, forthcoming). Other methods this analyst has used in the past to filter down data include: filtering by how many people tweeted search terms a certain number of times (e.g. 3 or more), filtering on word frequency and date, and filtering on geolocation. Once she has found interesting users, she reports turning to the Twitter website to examine user profiles and their contextual streams.

Analyst A3 reports using MySQL to extract a predetermined number of tweets per day that is suitable for annotation- generally around 2000 tweets/day. This extraction happens through random sampling. She then feeds these tweets to a classifier that determines if the tweet displays situational awareness or not. The classifier has yet to be tested on large datasets, and requires further optimization; but eventually she would like to input the entire dataset to the classifier.

Analyst A4 reports that her probable first step would be the identification of the date range of the event, then a filter based on dates with high volume of tweets. Then, she has developed a method of heavytail sampling: she randomly selects tweets from the set, takes the user that owns that tweet, puts that user in a separate set, and caps that at some number $n$ users (where $n$ is between 500 and 2000, depending on the event). Once she has an appropriately sized random sample, this sample is fed into eDataViewer, her tool that she has written for visualizing data (Starbird, 2007). eDataViewer groups data by user and requires $\leq 200,000$ tweets (or approximately

---

4 Contextual streams are explained in more detail in Section 4.1.2.2
5 See Section 5.2 for more details on this method.
2000 users). She reports that her methods of further analysis include using eDataViewer, Excel, and Google maps.

Analyst A5 reports that when faced with a new event, she will usually research it by looking at journalistic articles on the Web. She does this both to determine event details and to focus the research questions. Once she has obtained a reasonably sized dataset, she usually uses Excel and eDataViewer for visualization and analysis as well as further constraints and filtering. She also is interested in investigating third party Twitter applications; for example, one that reports how long an individual user has been on Twitter (How Long on Twitter, 2012).

3.2 Research practice findings

It is important to realize that each analyst’s reported research practice reflects work and methods that have proven useful and successful in the past. However, new datasets are expected to present unforeseen and different challenges that will probably require an additional, as yet unknown set of analytic approaches. Each analyst reports the usage of a variety of techniques as determined by the dataset. Thus this study is meant to be an indication of the different types of techniques brought to the table, more than just highlighting differences between analysts.

Each analyst does have different needs. However, they all have a similar requirement in that they must have a way to filter large datasets into something that is manageable. Prior to the analyst application, filtering was done either by hand or through database manipulation. Once a dataset was appropriately sized, the analysts then used tools such as Excel, eDataViewer, and searches on Twitter to further investigate the data.
3.3 Published research practices and related work

The individual interviews indicate that research practice focusing on Twitter communication has involved both tedious and difficult data manipulation and coding. To further illustrate some of the challenges, the methods of data collection for four papers are cited here.

For these efforts, the researchers obtained both search and filter data. Search data uses the Twitter Search API, and looks back in time for tweet matches, while filter data uses the Twitter Streaming API and collects off the Twitter firehose. One of the challenges of working in the emergency disaster space is the necessity of making rapid decisions about the nature of the communication, such as which hashtags people are using; and sometimes this is difficult to determine until after a term has gained popularity. So if a term is not discovered until later in the event, the Search API is used to collect back in time.

The paper by Starbird et al (2010) analyzes Twitter data that was collected over 51 days, from March 8, 2009, through April 27, 2009. Data collection was started soon after the onset of the flood threat period. For this event, the terms were ‘red river’ and ‘redriver’, and these hashtags were identified shortly after the event had occurred. A search was run back in time to retrieve hashtag data, resulting in 13,153 tweets and 4983 unique authors. Then the contextual stream was retrieved for each user in that sample. This yielded over 4.5 million tweets. Data analysis for this paper involved qualitatively examining and coding individual tweets and user tweet streams. The analysts used E-Data Viewer to visualize data sets, and read through hundred of user streams and

---

6 See Section 6.1 for more details.
thousands of messages. They read user biographies and profile pages, and traversed links in the tweets. The filtered data sets were generated through a process of coding individual tweets and authors as on or off topic. After the first pass, the data set was confined to tweets from users with more than three tweets on topic. This resulted in 7183 tweets. Finally, each researcher had to go and re-read each tweet to ensure correct coding.

The paper by Verma et al (2011) used data from four different datasets. The authors took random samples from each, extracting roughly 500 tweets per dataset. 500 tweets was selected as being a manageable amount for human annotation but also large enough to train a classifier. For each tweet, a determination was made as to whether the tweet was on- or off-topic. 527 tweets were sampled from the OK fire, and 453 tweets were sampled from the Red River flood; these tweets had been previously collected and sampled, and were known to be on-topic. The third dataset consisted of tweets collected on the 2010 Red River flood, and from the resulting sample of 500 tweets, only one tweet was off topic; resulting in 499 tweets. The Haiti dataset was also included, and 14 tweets of the 500 tweets randomly selected from this dataset were off-topic, leaving 486 tweets. The final dataset was 1,965 tweets. This was then all hand-annotated. Tweets were independently coded from each dataset by two annotators for four different qualities. These coded tweets were used for classifier training data.

Data collection for the paper by Starbird and Palen (2011) was a multi-step process. The analysts began with large-scale tweet data collection. The Twitter Search API was used to collect every tweet that contained both the #haiti hashtag and at least one specialized hashtag that corresponded to the Tweak the Tweet syntax. Then, every Twitterer was identified with a tweet in the dataset. The Twitter REST API was used to
capture each Twitterer’s contextual stream over the time period of interest. This yielded 339 Twitterers and a total of 292,928 tweets. Then, each contextual stream was analyzed, both for content and for a better understanding of how people were using the Tweak the Tweet syntax. Each tweet that contained a Tweak the Tweet hashtag was manually coded.

Finally, the paper by Sarcevic et al (2012) explains the painstaking, time consuming process necessary to extract 110 emergency medical response teams and organizations from the huge dataset collected after the Haiti earthquake. This dataset is comprised of 3.28 million tweets and 799,955 Twitterers. To filter down the 3.28 million tweets to the final dataset of 110 Twitterers and ~16,000 tweets, many techniques were used. First, the analysts searched only on the keyword ‘haiti’, which yielded 2.5 million tweets. From this, term frequencies were calculated for all words; also links, email addresses, numbers and single letters were removed. This generated over 200,000 terms, of which almost 100,000 occurred only once. Then only tweets that contained multiple-frequency terms were kept. This cut the dataset down to ~111,000 tweets. From these tweets, 1,686 medical terms were identified, and a search was re-run to find tweets that contained these terms. This generated ~300,000 tweets, which was used to construct a revised list of 1,544 medical terms. These terms were used on the original ‘haiti-keyword’ dataset to generate a medical-keywords dataset of 175k tweets and 89,000 Twitterers. Then, of this, Twitterers were selected with 10 or more medical term tweets, and 10% of these high volume Twitterers were sampled. This yielded 176 Twitterers and 9,675 tweets. These 176 Twitterers were analyzed, and 110 were identified as being on the ground.
From every example above, it is clear that the manual task of carefully filtering down a huge dataset is extremely time consuming. The iterative process of analyzing, filtering, sampling, and finally coding large datasets becomes nearly intractable as the datasets grow to be in the millions of tweets. While the manual work has been necessary to inform the software requirements, a software tool that allows the automation of much of this effort will be greatly beneficial from both an analysis and resources perspective.

4 Investigative user research

I conducted an investigative study to identify what is needed from the software to support the current research practice. This study involved both initial interviews and paper prototyping based on the interview results. I interviewed the five analysts associated with Project EPIC; as mentioned above, at the time of the investigative study, these five comprised the core analyst group. Each analyst has a specific set of requirements since each analyst conducts different research. The goal of the investigative study was to identify the separate research interests, elicit each analyst’s requirements, and come up with a general set of new requirements for the software that covers the common ground between the five researchers.

This investigative research is the first of the three components that make up the user study. In this component, each analyst was interviewed separately to determine the nature of his/her work, previous work practice, and what is needed in a new system. The intended outcome was requirements elicitation and analysis, resulting in a more complete set of requirements. The second component was the presentation of a mock UI to pairs of analysts for their evaluation and comment. The result was to get a better sense of how the tool should function when implemented. The third component, covered
later in Section 7, presented the first iteration of the tool and the UI to the analysts for comment and analysis of usability. The results of the first two components of the user study are as follows. For a discussion of the usability study conducted on the first iteration of the software, see Section 7.

4.1 Interview Questions, Answers, and Commonalities

Each question is stated, and is followed by the common findings amongst the analysts. Individual responses are also reported where they deviated from the commonalities.

4.1.1 General research interest

The question posed was, “What is your general research interest?”

4.1.1.1 Common Findings

The finding among this group is that everyone has a slightly different research interest within the space of crisis informatics. Thus it is expected that everyone will have different needs for data and analysis.

4.1.1.2 Individual Responses

A1 reports that her interests are reconnaissance and recovery building as they apply to the resilience of communities, especially as they happen after the emergency period of 3 weeks. She is particularly interested in looking at animals, pets, looting, and price gouging.

A2 studies how natural language processing can be integrated into human centered computing to help members of the public gain situational awareness in emergency.
A3 studies natural language processing and the development of classifiers.

A4 studies digital volunteerism during disasters; she also studies mass disruption events, including political protests.

A5 studies computer supported cooperative work, and is involved in any work supporting and involving Project EPIC. She is interested in how people do analysis, behavioral phenomenon, and Information and Communication Technology (ICT).

4.1.2 Dataset information

The question posed was, “What do you want to know about this dataset? For example, what would be the first thing you would look for?”

4.1.2.1 Common Findings

The following statistics represent the commonalities in responses to this question. All of the analysts agreed upon the utility of these statistics.

- Volume of tweets
- Date ranges of collected tweets
- What the search terms are and were
- How many unique users there were in the data set
- How many URLs there were in the data set
- What were the top 10 URLs
- What were the most retweeted tweets
- What percentage of tweets were retweets
- What percentage of tweets were modified tweets
Most analysts would like a graph of volume of tweets per time. This is useful in determining date range constraints. It can also potentially be useful for machine learning and event detection; certain high-volume time periods of tweets can be used as training data that show event occurrence.

4.1.2.2 Individual responses

In addition to the commonalities, other responses included wanting to have information about the following:

- User contextual streams: this is the tweet stream collected for a unique user in the dataset. It starts at the tweet of interest, and retrieves up to 3200 previous tweets, as limited by Twitter (Vieweg, 2012). A user contextual stream gives the reader a context for a particular tweet.
- Topic evolution over the event: what hashtags were popular at the event start, what were popular at the end, and how the popularity evolved; as well as when during the course of the event were they added as a filter
- Reporting of whether there was a need to search back in time to capture missing data. These back-in-time searches behave differently from filters (or the firehose capture7).
- Comparing events (i.e. comparing one dataset to others).
- Parsing in different languages where appropriate.
- Investigation of the nouns of the event, in order to formulate the right research questions.

---

7 See Section 6
- Geolocated tweets, and where those tweets are coming from; that is, if they are on the ground of the event. Often, a pertinent research question is to try to determine what is happening on the ground, so any information facilitating that is key.
- The ability to selectively follow; this is where a certain group of Twitterers are hand-selected, and known to be of interest. Then, filtering on Twitter streams can be limited to just tweets by and mentions of these particular users. Being able to select a certain group and explore it more would help in researching the question of how to look for problems perceived by the public, and how the public is self-organizing around solving those problems.

**4.1.3 Classifying a good dataset**

The question posed was, “How do you classify a good data set?”

*4.1.3.1 Common Findings*

A good dataset is predicated on good query terms. Good query terms are terms that are prevalent, trending, and capture the data associated with the event. Also, a good data set is one that has been collected over an appropriate length of time. The right query terms in a good dataset have been identified and introduced into the collection as they began trending on Twitter. Additionally, this dataset should have minimal, documented blackout periods.

*4.1.3.2 Individual responses*

Each analyst responded in a slightly different way, reflecting individual research interests. A1 would like a dataset with a lot of different content in it, such as multimedia content and external links. She also wants to see diversity in the types of people that are
talking about the event. This diversity can be expressed in terms of background, location, ethnicity, connection to the event, etc.

Analyst A2 is interested in user behavior; e.g. number of friends and followers at the beginning and end of an event. A good dataset will contain this information for interesting users.

Analyst A3 wants a dataset with as much data as possible. Her primary concern is about data being too filtered by the time it gets to her.

Analyst A4 classifies a good dataset as one that has caught most of the right terms for most of the time period over the early impact phase. There are several factors at work that contribute to this scenario. First, there is an expectation that the entire event is captured as long as the data collection was set up on time. Thus recording the time of data collection start is important. If event collection was started late, then there should be search data (that goes back in time) to capture as much of the beginning of the event as possible. Often, Twitter only releases a certain volume of search data, so it is possible that starting event collection late will result in an incomplete data set if it is a high volume event.

A4 and A5 both point out that a good dataset is one that has the appropriate date range, search terms, and users that can generate a series of snapshot analytics. These analytics should be representative of the event as a whole.

4.1.4 Dataset Usability

The question posed was, “What would it mean to you for this set to be usable?”

The common findings were that a usable data set is one with no (or minimal) gaps in collection time. A usable dataset is a good dataset, as elaborated on in the
previous section. Usability for all analysts is contingent on the ability to filter; e.g. filter on users, by locality, by date range, and to have no size restriction.

4.1.5 **Blackout periods**

The question posed was, “What about gaps in collection time and/or blackout periods?” The intention for this question was to determine both how big a deal blackout periods are and whether or not a dataset can still be useful with them.

4.1.5.1 **Common Findings**

All of the analysts require knowledge of the duration and temporal location of blackout periods.

4.1.5.2 **Individual Responses**

Blackout periods are a big issue for A1. Her research often requires following specific users and their activity as the event progresses. Blackout periods can make this impossible. A dataset can potentially be rendered unusable, depending on the duration and temporal location of the blackout periods.

For other analysts, blackout periods are manageable as long as there is knowledge of where and when they occurred. For A4, it depends on how big the gaps are; for example, a 15-minute blackout period is fine, but a blackout period ranging from 6 hours to a day renders the data less usable.

A5 wants the blackouts to be included in a visual display of volume of tweets per time while collecting. She would like to see blackout periods reported in the time zone of the event location.

4.1.6 **Requirements elicitation**
The question posed was, “What kind of help do you need to do the work you want to do?”

4.1.6.1 Common Findings

All of the analysts expressed a desire to filter the dataset in some way in order to make it more useful for research purposes. The answers to this question resulted in a series of filters that were incorporated into the final implementation, including:

- filter by date range
- filter by users: this facilitates picking out both interesting users and known digital volunteers (such as Humanity Road contributors, people who used Tweak the Tweet, etc).
- filter by retweets (both including and excluding)
- filter by search terms
- filter by number of tweets per user; for example, all users with more than 10 tweets, or all users with only 1 tweet.
- filter by geolocation and by geolocation bounding box

It was also important that the analyst could choose the search implementation. The original application was written using Lucene ranking (Lucene Scoring, 2012) to return relevant results. Lucene ranking calculates the document and term frequencies and employs cosine similarity. The analysts preferred non-Lucene ranked, Boolean search results; this means returning only terms that contain a certain search term ordered and ranked by date.
In addition to filtering, the analysts need to sample the data. This includes both sampling a given percentage of the original dataset and specifying a certain number of tweets for the final dataset, as well as a Twitter-based long tail sampling method.

Most analysts work in Excel at some point during the process. One of the analysts also works with MySQL and JSON formats. Thus it is important for the application to be able to export data in both CSV and JSON formats.

4.1.6.2 Individual responses

Again, each researcher had a slightly different take on this question. Analyst A1 reports wanting to watch and monitor an event as it happens, so she can flag interesting users and trends. Once she has noted these, she then wants to search and filter on them. She thinks she would probably filter just enough to get the data into Excel, her preferred work environment.

A3 needs programmatic access to the data. She issues her own MySQL queries and is comfortable in that environment, so having files in CSV or JSON formats are preferred. Her only filtering needs are by date range; after that, she would like to be able to export all of the data to read it into MySQL.

A4 has developed filtering and sampling methods unique to her own needs. She would like to see functionality that allows filters by the top $x$ retweeted people or top $x$ mentioned people. She also would like to be able to perform random samples of users by threshold, such as getting all tweets from 10% of all users with > 10 tweets.

A4 also wants to be able to specify a sample set of everyone who tweeted within a certain time range; e.g. 5% or 10%. Once she has filtered and sampled, she wants to have a visualization and histogram for everything; such as number of retweets (percentage
and volume), top \( x \) mentioned people/time, a histogram of people mentioned once through to people mentioned 500 times, and so on.

A5 needs to be able to have a way to investigate sub populations. It is most important to her to be able to parse up the space in the beginning, whether this is by date, or by language, or by whatever metric is most appropriate given the event. To help in making this decision, she wants to see statistics about the event, especially in real time, such as real time reporting on the percentage of tweets that are retweets, the percentage that are geolocated, or to see a distribution of who is the most high volume tweeter vs. low volume tweeters. For example, for the Japan event, one research question that interests her is how many people joined Twitter specifically to tweet about the event. Thus she would want to take some subset of tweets (e.g. all Japanese language tweets), figure out when the Twitterer of each tweet joined Twitter, get all of these dates, then sort on them to catalogue new joiners for an event.

Analyst A5 also expressed wanting to be able to save her filters and searches such that when a filtered dataset is written out, the method of obtaining that dataset is written out with it. Finally, she is hoping to be able to leverage something like Google translator for different languages.

### 4.1.7 Prior workflow

The question posed was, “What is your workflow?” This question has been covered by section 2.

### 4.1.8 Ideal software implementation

The question posed was, “What would be your ideal situation with 5 million tweets?”
4.1.8.1 Common Findings

The reported practices of the analysts combined with the challenges of working with a large dataset necessitate several different sections of the analyst application. First, the analysts need a webpage of descriptive statistics, both for a filtered event and unfiltered event. Second, they need filter functionality, presented on a webpage containing a series of filters that can be applied to an event. Third, they need a page for sampling. After each filtering or sampling operation is performed, a smaller dataset is returned that can then be exported to CSV or JSON. This overview captures the new requirements for the analyst application. These requirements are written up in more detail below in Section 5.

4.1.8.2 Individual Responses

When presented with 5 million tweets, A1 would likely browse through it to see what hashtags are emerging related to her research interest, and how they are emerging. She is more concerned with obtaining a representative sample across the whole event, so constraining by date range is not as important to her. However, she acknowledges that filtering by date range is one way of making the dataset small enough to fit into Excel. She expressed the need to be able to drill down into a dataset: to select certain users and be able to see just those users, or similar. Particularly, her method of drilling down is specific to the dataset, and she cannot know exactly what she needs until the data have been collected. She is looking for trends, and the way these trends are presented might differ from event to event. Thus, flexibility in the architecture is paramount.
A2’s probable first step would be to constrain by date range. To determine which dates are worth considering, she would use the graph depicting volume of tweets over time. Once she has a date range constraint, she would probably take a random sample within this date range in order to better investigate the data. She is usually looking for interesting Twitterers; what determines ‘interesting’ depends on the dataset. If she identifies interesting Twitterers, she will then most likely go back to the analyst application and filter on just those users.

Analyst A3 would like to be able to incorporate her classifier into the given framework. Since all of her work happens within MySQL, her ideal situation is to have programmatic access to the data.

Analyst A4 would probably approach this situation in a similar way to A2: she would first constrain by date range, and then she would take a random sample. She discussed the importance of visualizing results, such as percentage of retweets in the set over time, percentage of retweets per user, unique users over time, etc. She also wants to see a histogram of how many users had just one tweet in the set; i.e. a rendering of distribution of users over volume. This would be useful in determining if it is a good idea to cut by volume (e.g. cut users that have less than x tweets).

4.2 Mockup response

At the conclusion of the individual interviews, I created a paper prototype of a user interface, intended for analyst commentary. It included all of the common features mentioned in the interviews as well as many of the individual requests. I conducted two meetings, with two analysts in each meeting. Both interviews provided some feedback on interface design. All of the participants were generally satisfied with the mockup.
Once the interviews were complete, the team\textsuperscript{8} felt that a consensus was reached about the direction of the new software. The requirements were written up and recorded in Rally, our project management software tool.

The interview conclusion gave a good sense of reported work practices. These reported practices illustrate how all of the analysts use a variety of techniques for the task of investigating each dataset.

5 Evolution of requirements

The completion of the investigative research study allowed for an enumeration of the necessary tools and features important to the analysts. The software engineering team\textsuperscript{9} determined that the analyst application should be split up into several sections: descriptive statistics, filtering, sampling, histograms and graphs, classifier integration, and third party application integration. For the initial iteration of the software, we tackled only the first four.

The entire analyst application is based on the idea that the analysts are considering two different datasets at once: the full dataset, and the dataset with a number of filters and/or samples applied. This second dataset is referred to as the filtered dataset.

The first section is descriptive statistics, which contains a number of metrics describing both the full event data set and the current filtered data set.

The second section is filters. This section is composed of various means of filtering a dataset down to a more reasonable size; for example, by date range, by users, by searching, etc. In order to maintain what has been filtered and what has not, each

\textsuperscript{8} The five analysts interviewed and myself
\textsuperscript{9} Aaron Schram, Ken Anderson and myself
A tweet has a bit that indicates whether or not it is part of the current data subset. If the bit is flipped, the tweet is part of the filtered dataset.

The third section is sampling. This section allows the analysts to specify sampling rules and apply them to the filtered dataset to further whittle down the results. In addition to sampling by desired percentage or by desired final number of tweets, a specific long tail sampling method has been provided (pseudo code below in Figure 5.2).

The fourth section is graphs and histograms. This includes histograms plotting volume of users over time, as well as a graph depicting users and the percentage they contribute to the overall volume of tweets. Many more graphs and histograms were proposed, and should be addressed as future work.

The fifth section, included in the section on future work, is classifier integration. One of the analysts wrote a classifier (Verma et al, 2011) to make a determination between tweets that contain situational awareness and those that do not. Incorporating this classifier into the framework would present another filtering option for the analysts. Data sets could be fed into the classifier, and the analysts would then be able to work with data that has been classified as having situational awareness.

Finally, it is worthwhile to investigate the integration of third party applications. Third party applications are written by outside developers and deliver information relating to Twitter users and/or tweets. One example is an application that determines how long a user has been on Twitter.

### 5.1 Descriptive Statistics
The descriptive statistics web page provides the analysts with a series of calculated statistics about both the unfiltered event and the filtered dataset. This research determined that the following statistics would be supported:

- Total number of tweets in the dataset
- Date range of tweets contained in dataset
- The terms provided to the Twitter Streaming API that resulted in the collection of this dataset
- The number of unique users in this dataset
- The total number of URLs present in the tweet text of the dataset
- The percentage of tweets that contained URLs
- The total number of unique URLs present in the tweet text of the dataset
- Total number of retweets
- The percentage of tweets that are retweets
- The total number of modified tweets
- The percentage of tweets that are modified tweets
- All recorded downtime experienced during collection (holes in the dataset)
- The percentage of geolocation tweets within the dataset
- The distribution of language within the dataset

This page displays a table with two columns, one that contains unfiltered statistics and the other that contains filtered statistics.

5.2 Filters

The filters web page is the workhorse of the analyst application. It provides the analysts with a series of filters that cut the dataset down to manageable numbers. The
following operations should be supported on an individual event basis. Each operation is listed with an example underneath in italics.

- Date range constraint (constrained on tweet creation date provided by Twitter)
  
  o *E.g. only show tweets created during 9/20/2011-10/11/2011*

- Boolean geolocation constraint
  
  o *show only tweets with geolocation information*

- Username constraint
  
  o *show only tweets for users @aaron, @ken, @casey*

- Tweets/user threshold constraint
  
  o *show only tweets where the twitter user's total tweet volume is greater than 3 (within the collected dataset)*

- Boolean only retweets
  
  o *show only retweeted tweets*

- Boolean remove retweets
  
  o *show only tweets that have not been retweeted*

- Constrain results to top X most mentioned users
  
  o *show only the tweets of the top 10 most mentioned users*

- Constrain results to tweets with geolocation and geolocation within a specified bounding box
  
  o *show only tweets with -132.001,-124.00,100.023,-90.001*

- Constrain results by search terms contained in tweets
  
  o *show only tweets containing "walrus AND spaghetti"*

- Constrain results by terms not contained in tweets
- show only tweets not containing "requirements OR boring"
  - Allow for toggle between Lucene ranking of search results and Boolean ranking

The user study made it clear that the analysts prefer to *not* have Lucene rank and score their search results. Lucene is still used to index all of the tweets and to filter them on search execution\(^\text{10}\). However, when returning search results, the analysts want to have the option to return tweets that are a Boolean match of the search term, sorted by date. In contrast, Lucene scoring looks at term frequency and document frequency, and computes a cosine similarity based on these metrics to determine the best match to the query (Lucene Scoring, 2012).

### 5.3 Sampling

The sampling frame allows the analyst to sample the current data set in three ways. The analysts can either specify the number of tweets they would like returned; or, they can specify a percentage of the total number of tweets they would like returned. The sampling code then iterates over the tweets and pulls out every \(x\)th tweet sorted by date, where \(x\) is the interval between tweets that yields the desired total.

The sampling code also provides an option to implement a heavy tail distribution of Twitterers (Starbird et al, 2012 and Starbird, personal communication). To create a random sample of Twitterers that is generally representative of Tweet data, it is important to select from tweets rather than Twitterers. This is because tweet-volume per user has a heavytailed distribution favoring low volume users; a tweet-based sampling strategy flattens the distribution, and allows for a heavier sampling of higher volume

\(^{10}\) See Architecture, Section 6
users (Starbird et al, 2012). The way this works is as follows: first, a desired number of Twitterers is determined. Then, a tweet is selected at random from the data set, and the Twitterer that sent that tweet is added to the sample. This process is repeated until the desired number of Twitterers is obtained, discarding repeats.

```javascript
SelectTwitterers(numTwitterersWanted)
Twitterers = []
while Twitterers.length < numTwitterersWanted
    tweet = SelectRandomTweet()
    selectedTwitterer = GetTwittererFromTweet(tweet)
    if !Twitterers.hasItem(selectedTwitterer)
        Twitterers.push(selectedTwitterer)
}
return Twitterers
```

Figure 5.2 Heavytail sampling pseudo-code (Starbird, personal communication)

### 5.4 Graphical representations

The graphical representations currently implemented include volume of tweets over time and user contribution graphs. The user contribution graph shows a histogram of users vs. volume of tweets. The number of tweets is shown on the x-axis, and the number of users is shown on the y-axis. So, for example, one data point would be 3 users that tweeted 10 times.

### 5.5 Classifier incorporation

Sudha Verma authored a situational awareness classifier (Verma, personal communication) that has been trained on annotated situational awareness tweets. Integration would make the classifier available to run on tweets loaded into the analyst application. After the classifier runs, it will produce a subset of tweets tagged as having situational awareness.
5.6 Third party application integration

Incorporating third party applications will enhance the usability of the application. One such desired integration is with an application that determines how long a user has been on Twitter. This would be useful, for example, in answering questions such as how many users joined Twitter just to participate in a crisis event.

6. Evolution of Software Architecture

6.1 Data Collection Requirements and Overview

In order to be usable to the analysts, the data collection infrastructure has several key high-level requirements that must be met. First, it is required to be scalable, capable of collecting up to many millions of tweets per event. Disk space for these events generally runs in the hundreds of gigabytes. Second, the infrastructure must be concurrent, enabling high-volume data collection. All of the code is multithreaded and adaptable to multi core machines. Lastly, the infrastructure must be flexible, in order to adapt quickly to new requirements that arise during data collection. Often, emergency response means that new and unforeseen circumstances dictate different data collection, such as user profiles, friends and followers, etc. This flexibility must be built into the system.

Data collection happens on a per event basis. The software concept of an event contains all of the terms searched on pertaining to the actual, real-life event. These terms are represented in the software as constructs called TwitterQueries. There is a one-to-many relationship between each event and its TwitterQueries. The software collects tweets that contain each TwitterQuery, and there is a one-to-many relationship between each TwitterQuery and the corresponding tweets. To illustrate, an example of a
recent event for which data was collected was the earthquake that happened in Turkey in October of 2011. The TwitterQueries for the Turkey earthquake event are: ‘Turkey’, ‘van’, ‘Ercis’, ‘turkeyquake’, ‘Deprem’, ‘vanearthquake’, and ‘evimevindirVan’.

**Figure 6.1 Relationships between Event, TwitterQueries and Tweets.** Here, for example, the event is “Turkey Earthquake”. One TwitterQuery is #earthquake. Tweets can belong to different TwitterQueries if they contain more than one TwitterQuery term. In these instances, the tweets are stored twice. This is shown above, where “did you feel the earthquake #earthquake #turkey” is associated with both the #turkey and #earthquake TwitterQueries.

It is possible to duplicate tweets within an event, since a tweet can be collected that matches two TwitterQueries; for example, if a tweet has both ‘#earthquake and ‘#turkey in it, it will exist within the collection for both TwitterQueries ‘#earthquake’ and ‘#turkey’, both of which are associated with one event. See Figure 6.1 above. The architecture does not do bareword matching, meaning the full word must be matched for it to belong to a TwitterQuery.

A TwitterQuery can be one of two types: **filter** and **search**. TwitterQuery **filters** are created using the Twitter Streaming API. They collect data against the current Twitter firehose, and are also subject to any limitations Twitter imposes. For each Twitter filter, it is likely that there is data that is not delivered from the Twitter side due
to rate limitations. For the current infrastructure, all new TwitterQueries are created as Twitter filters.

*Searches go back in time, and are also subject to rate limitations as defined in the Twitter Search API.* How well a collection of tweets obtained by a search query represents a data set can be highly variable, depending on Twitter rate limitations, amount of time elapsed since the start of the event, and volume of traffic corresponding to that query. Twitter searches are intended to be stopgap measures to cover time periods between the start of the event (when a TwitterQuery might not be known) and the discovery of this appropriate TwitterQuery. Currently, searches do not flow through the main application. A new search needs to be run independently on the command line.

### 6.2 Software Architecture prior to User Study

Prior to the user study, a graduate student member of Project EPIC, Aaron Schram, had authored most of the web stack. The stack was primarily used for collection purposes. It also was responsible for displaying and editing events, TwitterQueries, and tweets associated with TwitterQueries. Finally, the stack allowed search operations within events.

The infrastructure is currently focused only on Twitter. It provides functionality that collects both tweets matching a given set of keywords and a subset of tweets generated by a given set of users. The code leverages Twitter’s APIs, including the Streaming API that captures tweets as they occur and the REST API that captures user information. The first incarnation of the software used the Search API to search back in time for relevant information (Anderson and Schram, 2011).
Spring (VMWare, 2012) is used to configure and manage database transactions. Spring was chosen as the best candidate to meet the data collection requirements stated above concerning scalability, concurrency, and flexibility. The web application is implemented using Spring MVC’s framework. A REST-based web service interface (Fielding, 2000) is implemented for each service; these “endpoints” are also managed by Spring. All of the model objects conform to JPA 2.0, with Hibernate (Apache Hibernate, 2012) as the JPA provider. Hibernate Search creates and maintains an index into the data model of services; it makes use of Lucene (Apache, 2012) underneath to actually maintain the index. The main advantage to Hibernate Search is that it lets the construction of the index occur in a concurrent manner; the index can be updated alongside inserts and updates of the underlying database (Anderson and Schram, 2011).

The architecture implements a layered service model of Domain, Persistence, Service and Application tiers. The Domain layer defines annotated domain objects used throughout the application; these include concepts such as Tweet and TwitterQuery. The persistence tier, which uses Hibernate as the object relational manager, manages all transactions and persistence operations. The transaction framework uses the entity manager in a non-traditional way such that shared objects are held for the smallest amount of time possible for the facilitation of better concurrency. The Service tier is where all domain-specific services live; it interacts heavily with persistence and the transactional capabilities. Services communicate with a MySQL database using the Java Persistence Query Language. Services are dependency injected by Spring as needed into Controllers, which define RESTful endpoints on the server side. These endpoints are accessed asynchronously from client-side Javascript contained in a Java Server Page.
(JSP). The application tier accesses domain objects via the services tier without having to write persistence code. For more details, please see Anderson and Schram (2011).

### 6.2.1 Personal contributions to the old architecture

Personal contributions to the old architecture were largely concentrated on interface tasks for event management and search implementation. I completed four main tasks: search, a home page tree view, editing of events and editing of TwitterQueries.

### 6.2.2 Search

Aaron Schram and I collaborated on the implementation of the back-end search functionality. I was responsible for the front-end display. On the old architecture, search is implemented with Lucene. The text of each tweet is indexed using the Lucene StandardAnalyzer (v3.0.1) (Apache Lucene StandardAnalyzer, 2012). This implements stemming of search terms for better search matching. Queries are constructed using a MultiFieldQueryProcessor and passed to a FullTextQueryBuilder. This builder can take parameters such as filters and maxResults. The search service is then able to use the builder to perform the search, and is also responsible for paging the results in sets of 200. These results are returned to the JSP that renders them into a YUI2 DataTable (also paged). Search syntax supports all Lucene query language functionality.

![Figure 6.2 Architecture interaction with Lucene](image)

---

Figure 6.2 Architecture interaction with Lucene
Figure 6.3 Screenshot of search on epic-collect version 1. Various fields are available for selection to view in the table below. Results are displayed in the table at the bottom of the page. Also available are options to export the data.

6.2.3 Home page tree view

I created an index page to display all events and their corresponding TwitterQueries in a tree view. Options to display inactive events were included.
Figure 6.4 **Home page tree view.** All events are listed, with TwitterQueries as event children. Inactive events are greyed out.

Clicking on an event or TwitterQuery will take the user to the event edit page.

### 6.2.4 Event and TwitterQuery Editing

I created an editing screen to allow for editing of events and TwitterQueries.

Security features prevent editing unless a user logs in as an admin.
A JSP queries the server for all TwitterQueries associated with the event the user wishes to edit. All desired edits are first done on the client side. The user can edit the event name and disable the event. The user can also add TwitterQuery filters, delete them, rename them, or reassign them to a different event. Once the user is done editing the event and its TwitterQueries on the client side, the edits are submitted to an endpoint on the server, which correspondingly updates the database.

On this page, the user can create new TwitterQueries for an event. All new TwitterQueries created are Twitter filters collecting directly from Twitter’s Streaming API. New Twitter searches cannot be created anymore, since, as explained above, filters now subsume search functionality. However, existing Twitter searches can be deleted or rendered inactive.

6.3 Software Architecture post User Study
Following the completion of the user study, the software engineering team determined that the old architecture was no longer sufficient to satisfy the new requirements. The new application would need to be split between analysis and collection. Additionally, collection would need to be moved to a Cassandra/NoSQL solution to prevent data loss and increase performance. Cassandra with NoSQL allows for dynamically varying columns, horizontal scalability, and distributed storage (Lakshman and Malik, 2010). A large number of cluster nodes can be maintained and written to in parallel.

Designing and implementing the new architecture was apportioned between Aaron Schram and myself. We maintained the sense of the original structure, with the stack divided up into model, persistence, and service tiers (Figure 6.7); but much of the code was changed or removed within those tiers. Underneath the Webapp component, the two applications are now ‘search’ and ‘analyst’, and the analyst directory contains all components related to the analyst application. The new workflow dictates that the collection architecture will run while an event is happening. This collection architecture leverages Cassandra and NoSQL. Once the event is determined to be finished\(^\text{11}\), the database of tweets for that event can be transferred to a MySQL database. This is sent to the analyst application, which runs on one MySQL database instance at a time. See Figures 6.8 – 6.10.

\(^{11}\) The analyst team makes the call as to when an event is over.
**Figure 6.7 The analyst application tiered structure.** Model is used throughout. The top level is App and Webapp, where App contains binaries. The bottom tier is JPA, Hibernate, and raw Lucene.

**Figure 6.8 Architecture interaction between Cassandra collection software and analyst application - collection**
1. A user tweets “tweet”, which is sent to Twitter.
2. Tracking software running on EPIC-collect picks up this tweet’s existence.
3. The tweet is then sent to EPIC-collect from Twitter.
4. The tweet is sent to Cassandra to be stored.
Figure 6.9 Architecture interaction between Cassandra collection software and analyst application – storing tweets.

1. When the event is over, the export command is sent to EPIC-collect to get all tweets with the relevant TwitterQuery from Cassandra.
2. The browser endpoint creates the new MySQL database.
3. EPIC-collect populates the MySQL database with the relevant tweets.

Figure 6.10 Architecture interaction between Cassandra collection software and analyst application – analyst database access. The analyst application now has access to this database. It indexes all tweets in Lucene. The analyst application can serve browser requests.
The collection architecture is written and managed entirely by Aaron Schram. The analyst application is my responsibility. It uses Hibernate as the object relational model, JPA 2.0 annotations, Lucene for tweet indexing, and a MySQL back end. The structure of the analyst application models the architecture described in Section 6.1 (see Figure 6.9). The analyst application implements statistics, filtering, and sampling services, as described in the evolution of requirements above. It also provides the option to toggle off Lucene search ranking.

Figure 6.11 The new analyst application architecture follows a similar architectural diagram as the old architecture. The only component that I needed to develop was the analyst web application and some of the services. I relied on the EPIC persistence, model and database tiers that were already in place.
The MySQL database accessed by the analyst application has three tables: Tweet, TwitterQuery, and Mention. See Tables 6.1 – 6.3 for a description of the attributes on these tables.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createdAt</td>
<td>Date</td>
<td>Date the tweet was created as supplied by Twitter</td>
</tr>
<tr>
<td>latitude</td>
<td>Double</td>
<td>Latitude, if it exists</td>
</tr>
<tr>
<td>longitude</td>
<td>Double</td>
<td>Longitude, if it exists</td>
</tr>
<tr>
<td>dataSubset</td>
<td>Integer</td>
<td>Indicates if this tweet is part of the filtered dataset</td>
</tr>
<tr>
<td>retweetCount</td>
<td>Integer</td>
<td>Number of retweets for this tweet</td>
</tr>
<tr>
<td>uuid</td>
<td>Long</td>
<td>Twitter user’s unique user ID</td>
</tr>
<tr>
<td>language</td>
<td>String</td>
<td>User-reported language encoding</td>
</tr>
<tr>
<td>raw</td>
<td>String</td>
<td>The raw tweet in full</td>
</tr>
<tr>
<td>screenName</td>
<td>String</td>
<td>User’s screen name</td>
</tr>
<tr>
<td>text</td>
<td>String</td>
<td>Tweet text</td>
</tr>
<tr>
<td>twitterQuery</td>
<td>TwitterQuery</td>
<td>The twitterQuery that generated the collection of this tweet</td>
</tr>
</tbody>
</table>

**Table 6.1 Tweet attributes stored in MySQL.** The attributes on Tweet do not represent all possible attributes supplied by Twitter. However, the entire raw tweet is stored in case additional information is required. For example, the statistics page displays a list of unique URLs. The URL information is encoded deep in the raw tweet, which is extracted by the software.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createDate</td>
<td>Date</td>
<td>Creation date of the mention</td>
</tr>
<tr>
<td>tweet</td>
<td>Tweet</td>
<td>The tweet that contains this mention</td>
</tr>
<tr>
<td>username</td>
<td>String</td>
<td>The username that was mentioned (@mention)</td>
</tr>
</tbody>
</table>

**Table 6.2 Mention attributes stored in MySQL**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createDate</td>
<td>Date</td>
<td>Creation date of the TwitterQuery</td>
</tr>
<tr>
<td>query</td>
<td>String</td>
<td>The query (e.g. #earthquake)</td>
</tr>
</tbody>
</table>

**Table 6.3 TwitterQuery attributes stored in MySQL**

Tweet holds on to a reference to a TwitterQuery. The relationship between TwitterQuery and Tweet is one-to-many. Mention holds on to a Tweet. The relationship between Tweet and Mention is zero-to-many. See Figure 6.10.
Because the analysts wish to perform a series of filtering operations to reduce the dataset into a more manageable size, there is a need to maintain a set of tweets that represents the result of the applied filters. The software implements this by keeping a bit on each row in the Tweet table (in Table 6.1, this is dataSubset) that indicates whether or not the tweet is part of a subset. Every time the analyst executes a filter, the bits are updated to reflect the new filtered subset. When the database is first loaded, the dataset and the filtered dataset are the same, since no filters have yet been applied. An initial filter operation can be very expensive, since the subset is as large as the entire database; so every row in the database must be examined. Once a few filters have been performed, the number of rows requiring examination drops and performance increases.
Figure 6.13 Maintaining filtered datasets. The analyst executes filters on the current filtered dataset, which returns a series of matching tweets. The update function gets all tweets with their dataSubset bit flipped to 1 (meaning they are part of the current filtered dataset before the new filters get applied). Iterate over all tweets with the bit set to 1 and ask if the new list of filtered tweets contains the old tweet. If so, do nothing. Otherwise, flip this bit to 0, and update MySQL and Lucene.

Every part of the analyst application leverages Spring MVC attributes such as: dependency injection; model, view, and controller interactions; and JPA 2 annotations. Each section has one to many controllers. These controllers are dependency injected with services, and provide a series of RESTful endpoints used by the view.

6.3.1 Statistics

This page displays the necessary descriptive statistics determined in the user study about both the overall event and the filtered event. The statistics code contains an AnalyticsAPIController and an AnalyticsService. AnalyticsAPIController exposes a series of RESTful endpoints that provide statistical information for both filtered and unfiltered event-based datasets. It is dependency injected with AnalyticsService. AnalyticsService provides the transactions that communicate with the database via the Java Persistence Query Language (JPQL).
This page displays all of the filters described by the new requirements (Section 5.2) from the user study. It allows user filtering of data. The filtering code also provides search functionality and manages Lucene interaction. The filtering code contains a FilterAPIController, which provides a series of RESTful endpoints that expose filter operations. The FilterAPIController is dependency injected with both TweetService and FilterService. FilterService provides the transactions that communicate with the
database via JPQL. TweetService is then used to update the resulting filtered dataset by flipping the bits on the tweets. See Figure 6.12.

The FilterAPIController has one main method responsible for filtering. It executes most filter operations by concatenating JPQL statements and issuing one large query. Two queries need to be run independently. First, the query asking for top x
mentioned users has to be run separately, as it operates on the Mention table. Additionally, Lucene searches must be run after the JPQL query returns, since they require querying the Lucene index, which does not involve MySQL.

6.3.3 Samples

This page displays all of the samples described by the new requirements (Section 5.3) from the user study. It allows user sampling of data. The sampling portion of the analyst application contains a SampleAPIController; this provides a series of RESTful endpoints that expose sampling operations. The SampleAPIController is injected with the SampleService and the TweetService. SampleService is responsible for issuing JPQL queries to MySQL that return the desired list of sampled tweets. TweetService is responsible for updating the bits on the returned tweets to appropriately define the subset.

Figure 6.16 Sampling page screenshot

6.3.4 Histograms
This page displays all of the graphs described by the new requirements (Section 5.4) from the user study. It allows user graphing of data. The graphing portion of the analyst application contains a GraphAPIController; this provides a series of RESTful endpoints that expose graphing operations. The GraphAPIController is injected with GraphService and AnalyticsService.

To display volume of tweets over time, GraphService processes tweets in pages of 200 to ensure a small memory footprint. These tweets are then bucketed according to the number of tweets that occur within a certain interval. Once the processor has processed all tweets, a HashMap is returned containing the histogram information.

To display users and percentage of contribution to overall volume, first all users are obtained. Then, for each user, the number of tweets is queried. A sorted HashMap is returned mapping username to their volume of tweets.

The graphs are displayed using YUI 2 charting libraries.
Figure 6.17 Graph of volume of tweets over time. A blackout period can be seen towards the end of the dataset.

Figure 6.18 Graph of users and volume. 1 user has tweeted 33 times, 3 users have tweeted 11 times, 173 users have tweeted 3 times, etc.
6.2.5 Exporting

Exporting to CSV and JSON makes use of the ExportAPIController, which exposes a RESTful endpoint that handles the exporting. ExportAPIController is dependency injected with both ExportJSONService and ExportCSVService. When the endpoint is hit, the type of export desired is sent in on the request and the appropriate service call is made. Each export service processes tweets 200 at a time to reduce memory footprint. As the 200 tweets are processed, they are written out to the appropriate format. Then, the next 200 tweets are retrieved, processed, and written out.

7 Findings

7.1 Usability Study

After completing the initial set of requirements, I conducted a usability study on the first implementation of the software. I used 6 analysts, since that included everyone within Project EPIC that would possibly be interested in Twitter research and that would have use for the application. This group of 6 analysts includes 4 of the analysts from the investigative interviews and 2 analysts who were either new to the group or were willing to help with the research effort. To best approximate a real work environment, the usability study was done on tweets from a disaster dataset.

The usability study was conducted using search tweets (e.g. back-in-time tweets) from the San Diego power outage. For this data set, I chose search tweets over filter tweets as filter tweets were left running on the term “power” for over three months, and the results were intermingled with other data collection results; thus extracting the relevant data would have proved prohibitive given the time frame. The search tweets had already been collected, and totaled around 200,000. Of that 200,000, the final set
was cut down to 44,354; this was done by taking roughly the first 21,000 tweets collected on ‘power’, the first 21,000 collected on ‘outage’, and the entire ‘sdoutage’ result set of 1611. This data did not cover the entire San Diego power outage event. However, the purposes of the study were to (1) allow the analysts to iterate quickly enough to test out all the functionality in 30 minutes (which would have not been possible with 200,000 tweets) and (2) to give the user a general sense of the data. Since each tester would not be required to do further analysis beyond the user study, I decided that an incomplete data set would still serve these purposes.

Each user was given the analyst application loaded with the San Diego dataset. Each was told to work with the tool in the way in which they would approach any new research problem. They were encouraged to ask the research questions of particular interest to them. The study was intended to be open ended and tailored to each analyst’s specific workflow.

### 7.2 Usability Study Findings

Each analyst approached the tool slightly differently. This was helpful in that it uncovered use cases not previously considered. The process taken by each analyst is reported. Commonalities and additional requirements are also discussed.

#### 7.2.1 Analyst process

Analyst A1’s initial steps were to search on specific terms, using both Lucene and Boolean methods of ranking. Her choice of Lucene was unexpected, as initially every analyst expressed not being in favor of this method of ranking results. Her workflow consisted of using the search functionality coupled with the exclusion of retweets. After
these filtering steps, she stopped her investigation. However, from there, she would likely move to Excel to dive deeper.

Analyst A2 usually begins by doing research on the web to gather basic facts; when the event happened, how many people were affected, etc. This data informs the rest of her searches. She began the user study assuming she had already done this step. First, she filtered by excluding retweets, and read through the returned data set. Then, she did a search excluding all links from the dataset (anything containing http*). In scanning that data set, she noted that there was a lot of personal information returned, advice tweets, and tweets with subject deletion. She scanned the tweets looking for situational awareness information, such as people discussing particular areas hit by the power outage, homeless shelters, and crime. Finally, she ran a sample filter to end up with around 5000 tweets that could be given to annotaters. Were she to continue examining this dataset, her process would involve reading tweets to get a sense of how to make the next filter decision. For this dataset, she would have pursued how people are personally dealing with the outage on Twitter, and she would have done this by using search terms.

Analyst A3 was excluded from this study, as she will most likely not be using the tool.

Analyst A4 would have started the process by filtering on number of tweets per user. At the time of the user test, this particular filter was too slow to use\textsuperscript{12}. Her next step was to search on the word ‘outage’. When the results came back, she determined that there were too many; so she executed a geolocation filter with a smaller date range, using 9/7/2011 to 9/9/2011. Finally, she created a geolocation bounding box. She

\textsuperscript{12} This has since been fixed.
opened another tab with a Google map on it, located San Diego and the appropriate latitude and longitude measurements. She plugged these into the application and read through the resulting tweets.

Analyst A5 began with pen and paper. She loaded up the statistics page, and began recording information. She wrote down the duration of the data set. She looked at retweet and modified tweet statistics. She was also especially interested in language distribution: where each tweet was coming from and who was doing it. She noted what she would investigate further, such as certain language distribution percentages. She also had the initial thought that geolocation is of less importance to her per event, and that perhaps she is more concerned with geolocation across events.

After an initial glance at statistics, she continued her research on Google, noting potentially interesting research questions as she scanned articles. While doing Google research, she noted that one frequently relevant research question is the verification of journalistic reporting: sometimes there is a dichotomy between what is reported in the media and what is being discussed on the ground.

A5 then turned to the filters page. She determined that she first wanted to see all 44k tweets in Excel, so she loaded the data set and exported it. She went through Excel trying to get right to the edge of the event to determine start and end dates. Contrary to her first thought, she found that she did want to pursue geolocated tweets within the event; she noted that a few users had geotagged tweets, so she returned to the analyst application and exported all geotagged tweets. She brought this up in Excel, and suggested plotting latitude and longitude on a graph to visualize user location. She then went back to the full set of tweets in Excel and sorted by username. Ultimately, she wanted a histogram showing users and their number of tweets.
Her workflow for this small test event consisted of moving between Excel and the analyst app, going back and forth between each representation and filtering, then exporting. She imagined having several Excel files open at once that would be able to offer different windows on the data. She wants to be able to perform statistical and qualitative analysis simultaneously to determine where the biggest issues are, what they are, and how to capture them by narrowing the data set.

Analyst A6 did not have a specific research question in mind. Therefore, he began by searching for a ‘food’ term and retrieved results. From there, he submitted date filtering.

Analyst A7 began by searching on ‘*outage’. She then excluded retweets. Then she reset and searched again on ‘outage’ and ‘urgent’. Since she has not yet had the opportunity to do a lot of deep analysis of Twitter data, she focused more on UI direction and suggestions.

7.2.2 Synthesis of Usability Study

Analysts tended to begin either by searching on specific terms or using filters such as retweet exclusion and geolocation. The search filter was used more heavily than expected; most analysts turned to it or mentioned it at some point. This was unusual only because during the investigative report, the analysts did not place heavy emphasis on search. Also, the workflow of moving between the analyst application and Excel was repeated several times.

Beyond this point, analyst work practice diverged depending on specific research questions. The usability study resulted in additional requirements, detailed below.

7.2.3 User study additional requirements
This user testing resulted in several additional suggestions and requirements:

- Most of the analysts had UI suggestions, and one analyst provided a redesign of both the filter and sampling pages that aided in clarity and consistency.

- One analyst suggested a redesign of the filtering submission. Originally, date filters and other filters were separated. His proposal of combining the two seemed much more clear.

- With respect to search, the analysts asked for help on the web page about how to use both Lucene and Boolean searches. They specifically requested query language tutorials.

- Analyst A2 wants incorporation of the situational awareness classifier. She would use this as a primary filtering step once she had a data set of around 20,000 tweets.

- Filtering by number of tweets per user was too slow to use. An additional requirement is to investigate this filter and improve behavior (which has since been fixed).

- Several analysts did not think the workflow was intuitive. The design of the application was as a successive filtering tool; so the starting point is the whole data set, and the end result is a filtered data set. However, if a filter is executed and unwanted, it was unclear when and how to restore the original data set after filtering. At the very least, there should be a way of either alerting the user or somehow making the state of the filtered dataset more obvious.
- One analyst would like to see the percentage each user with a geotagged tweet contributed to the total volume of tweets.

Several analysts mentioned wanting to see a graph of volume of tweets vs. time, to better determine how to cut down by date range and to better determine peak days. This feature has since been implemented. Once it was deployed, one of the analysts provided more feedback about additional graph features she would like to see. In particular, she would like to be able to set the time interval between bars on the graph and customize some display options.

Additionally, analysts wanted to see histograms plotting the number of Twitterers with the number of tweets over the event (and percentage of the total). This also has since been implemented.

7.3 Performance

Performance is probably the single largest issue with the analyst application. Even with 44,000 tweets, the application can take anywhere from 10 seconds to minutes to complete certain filters. Clearly, this is problematic as some of the larger data sets have up to 9 or 10 million tweets.

The filters are issued as JPQL queries against a MySQL dataset. When a filter is issued against the whole dataset, there is nothing that can be done to avoid looking at every single row. Additionally, the way we are representing the filtered dataset in the software is by flipping a bit on each tweet that fits the new filtered requirements. When the filters are executed, the code must then iterate through the tweets and update the flipped bits to reflect the new filtered dataset. This also requires looking at every tweet
in the dataset that has a bit set to 1: first, all rows set as 1 are set to 0, then each row that fits the new filtered dataset is re-set to 1.

Flipping the bits and updating the dataset is one of the more expensive operations, as the Lucene index per tweet must also be updated. To facilitate a faster workflow, a button was added at the bottom of the application called ‘Estimate Reduction’. This button submits the query but does not update the Lucene index or database, and does not filter. It allows the user to see how many tweets will result from executing a filter (or series of filters) without actually executing it. The user can then make the determination if it is worth applying certain filters. Estimating the reduction is quite a bit faster for all queries. Note also that times will vary per operation per dataset. This is because the larger the dataset returned from the filter, the longer it will take to iterate through the dataset and update every bit.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Estimate Reduction</th>
<th>Submit Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top 4 most mentioned:</td>
<td>453 ms</td>
<td>897 ms</td>
</tr>
<tr>
<td>Load whole dataset:</td>
<td>3332 ms</td>
<td>106089 ms</td>
</tr>
<tr>
<td>Show only retweets:</td>
<td>2438 ms</td>
<td>60133 ms</td>
</tr>
<tr>
<td>Exclude retweets:</td>
<td>2110 ms</td>
<td>95055 ms</td>
</tr>
<tr>
<td>Show only geolocated:</td>
<td>480 ms</td>
<td>30107 ms</td>
</tr>
<tr>
<td>Filter by users (3):</td>
<td>441 ms</td>
<td>29678 ms</td>
</tr>
<tr>
<td>Filter by search terms (3):</td>
<td>657 ms</td>
<td>31480 ms</td>
</tr>
<tr>
<td>Filter by search terms and excluding retweets</td>
<td>674 ms</td>
<td>30702 ms</td>
</tr>
</tbody>
</table>

Table 7.1: Difference between Estimate Reduction and Submit Filter Operations. Metrics demonstrating the difference in time between the ‘Estimate Reduction’ operation and ‘Submit query’ operation (which updates the Lucene index and the database). The difference between the two operations is an update of the filtered dataset. This table shows how expensive it is to scan the rows and flip the necessary bits. Note that the specific times are less important than the difference between times for the operations. The specific times are a function of many factors including laptop processing speed, memory, MySQL speed, application speed, other programs that were open, etc.

One obvious performance increase would come from a Hadoop/MapReduce solution. This would involve migrating all of the data to a Hadoop Distributed File
System (HDFS), writing map jobs for each operation, writing reduce jobs, and distributing the load across a cluster of machines. See section 8.2 below.

8 Future work

Six components of the initial user study were not implemented due to time constraints. These were additional data visualizations, classifier incorporation, third party application integration, the ability to save filters, UI improvements, and switching between events.

The analysts suggested many data visualizations that could be beneficial to crisis informatics research. Due to time constraints, the only visualizations that were implemented were a histogram of volume of tweets over time and a histogram showing users vs. the volume of tweets per user. The focus of the remaining time was on implementing other requirements and investigating performance with big data. However, the opportunity to implement other visualizations would prove beneficial to all of the analysts.

Additionally, the situational awareness classifier would provide another method of filtering down data. The first step would be incorporation into the infrastructure such that the classifier can take the results of a filtered dataset as input. Then, the application would display as a result all tweets classified as having situational awareness. Performance remains the large unknown in this scenario.

Third, opportunities exist to integrate third party applications into the analyst application. What would be needed is to determine how to programmatically pass data to the third party application from the analyst application and retrieve the results.
Fourth, a component that came up during interviews was the idea of saving filter operations; that is, as an analyst applies filters, the system would know about the operations being performed and would save this out along with the data. Then, a record would be kept mapping filter operations to filtered datasets stored on disk. When the application loads, the user would have the option of either loading in a filtered dataset or viewing the set of filters and samples that have been previously performed.

Fifth, one of the analysts, Lise St. Denis, provided a UI redesign. Due to time constraints, it has not yet been implemented. This redesign will aid in the usability of the application.

Finally, the ability to switch between events in the analyst application should be implemented. Ideally, the box that runs the analyst application will also host many disaster datasets stored as MySQL files. The analyst should have an automated, intuitive way to choose which event is being examined.

8.1 Performance

The clearest way to improve performance of the system is to migrate the tasks to a distributed problem solving architecture such as Map/Reduce (see Apache Hadoop’s implementation, Apache Hadoop 2012). This would involve moving the data to a Hadoop Distributed File System (HDFS). It would then require partitioning the work into separate map and reduce jobs. The work involved in making this change would be substantial; every task would need to be rewritten as a series of map and reduce operations, and transferring the data to HDFS would involve the work of setting up the Hadoop cluster. The difficulty would be in setting up the HDFS system, partitioning the
work into reasonably sized chunks, writing the map and reduce jobs, and recombining the solutions (Dean and Ghemawat, 2004).

However, such an implementation will most likely be the only way the application functions on large datasets at all. Memory and execution time issues will almost certainly make running the analyst application on a large dataset impossible. Issuing a JPQL query on millions of rows on a MySQL database is prohibitively expensive. Additionally, this type of work is ideal for a map/reduce solution. None of the data solutions depend on each other: when a user runs a filter, all that needs to happen is that every tweet must be filtered. Thus the data could be partitioned into chunks and sent to map/reduce jobs to be processed in parallel. This is the most important component for future work and will be the primary factor determining the usability of the system.

9 Conclusion

With this thesis, I present tools and services in support of data analysis as it pertains to crisis informatics. The current state of the software is useful only for smaller data sets. As stated above, the software is not ready to be run on a dataset with millions of tweets. However, the current implementation is proof of concept that the analyst application will provide utility to the analysts as they pursue research in this space. Once the distributed solution is in place, this tool should prove indispensable for large-scale crisis informatics data analysis.
Bibliography


