Journal of Web Engineering, Vol. 16, No. 5&6 (2017) 399–432
© Rinton Press

GETTING THE QUERY RIGHT FOR CRISIS INFORMATICS DESIGN ISSUES FOR WEB-BASED ANALYSIS ENVIRONMENTS

MARIO BARRENECHEA, SAHAR JAMBI, AHMET ARIF AYDIN
MAZIN HAKEEM, KENNETH M. ANDERSON
Department of Computer Science, University of Colorado Boulder, 430 UCB
Boulder, Colorado 80309-0430, USA
{*mario.barrenechea,sahar.jambi,ahmet.aydin,mazin.hakeem,ken.anderson*}*@colorado.edu*

Web-based data analysis environments are powerful platforms for exploring large data sets. To ensure that these environments meet the needs of analysts, a human-centered design perspective is needed. Interfaces to these platforms should provide flexible search, support user-generated content, and enable collaboration. We report on our efforts to design and develop a web interface for a custom analytics platform—EPIC Analyze—which provides interactive search over large Twitter data sets collected during crisis events. We performed seven think-aloud sessions with researchers who regularly analyze crisis data sets and compiled their feedback. They identified a need for a "big picture" view of an event, flexible querying capabilities, and user-defined coding schemes. Adding these features allowed EPIC Analyze to meet the needs of these analysts and enable exploratory research on crisis data. In performing this work, we identified an opportunity to migrate the software architecture of EPIC Analyze to one based on microservices. We report on the lessons learned in performing this migration and the impact it had on EPIC Analyze's capabilities. We also reflect on the benefits a microservices approach can have on the design of data-intensive software systems like EPIC Analyze.

Keywords: data-intensive systems, web applications, design issues, crisis informatics

1 Introduction

We live in an era of big data. Our ability to generate and collect large amounts of data is having a transformative effect on the types of analysis we can perform. The term "big data" refers to a variety of techniques and technologies that enable this transformation and enable the creation of *data-intensive software systems.* These systems must collect, store, index, analyze, and annotate large sets of data, and there are significant challenges in making these systems scalable, reliable, and efficient [1]. Another set of challenges exist with respect to designing the user interface of these systems. These interfaces must provide users with a sense of scale, present details on demand, provide overviews, and provide a flexible set of operations that execute at interactive speeds.

We work in an application domain known as *crisis informatics* [2]; crisis informatics is a multidisciplinary research area that examines the socio-technical relationships among people, information, and technology during crisis events. It mainly examines the qualitative and quantitative aspects of social media data produced by members of the public during times of mass emergency. Our project—Project EPIC—has been collecting crisis data sets from Twitter since Fall 2009; we have now amassed approximately 2.5B tweets across hundreds of events [3, 4, 5]. As a result, we have been designing and developing a data analysis environment—EPIC Analyze [6]—that provides a variety of services to help Project EPIC analysts explore and understand our large Twitter data sets.

As a result, we have been wrestling with a number of thorny issues related to the design of data-intensive systems and their user interfaces [7]. In this paper, we report on the challenges we have encountered with designing user interfaces for services that enable the browsing, filtering, and annotation of large crisis data sets. For these services, our goals have been to a) provide interactive response times, to b) make it easy to query, filter, and explore a large data set, to c) ease the management and filtering of user-defined data, and to d) provide collaboration capabilities for our users. Drawing on techniques from human-centered computing, software engineering, and web engineering, our goal is to simplify the access to large crisis data sets and provide capabilities that allow EPIC Analyze to function as a vehicle for exploratory research on crisis data. Note: we do not discuss in this paper whether the data in these datasets are reliable or useful; our collaborators and colleagues in crisis informatics have tackled such issues in detail [8, 9, 10, 11]. Instead, we examine the types of design and implementation decisions that need to be made to make web-based analysis environments, like EPIC Analyze, useful to the analysts that rely on them. To this end, we performed think-aloud sessions with seven researchers who regularly analyze crisis data sets and compiled their feedback; this feedback drove the creation of the current version of EPIC Analyze.

This paper is organized as follows. In Section 2, we provide background information on the area of crisis informatics and discuss some of the challenges encountered when working with social media data. In Section 3, we present the the results of a user study on a previous version of EPIC Analyze that identified the enhancements that would be needed to make it an effective web-based analysis environment for crisis informatics research. In Section 4, we describe the data models and services that had to be added to EPIC Analyze to support the changes our analysts requested in the user study. In Section 5, we present the user interface and services of the resulting version of EPIC Analyze and discuss how it now meets the needs of our analysts. Section 6 introduces new design decisions that we have made recently to migrate EPIC Analyze to an architecture centered around microservices. In Section 7, we situate our work with respect to related research. Finally, we present avenues for future work in Section 8 and our conclusions in Section 9.

This paper is an extended version of a previous paper on EPIC Analyze [12]. This version is significantly restructed and contains expanded material with respect to our discussion of related work and our presentation of EPIC Analyze. In addition, Section 6 is completely new work that did not appear in the previous version and describes a new version of EPIC Analyze's software architecture based on microservices. To clarify, three separate versions of EPIC Analyze are discussed in this paper. The "current version" is presented in Section 5; this is the version of EPIC Analyze that is deployed for use by Project EPIC analysts. The "previous version" is discussed in Section 3; this version of EPIC Analyze was the version that was used in our user study. Finally, the "next version" of EPIC Analyze is the one described in Section 6 which is based on our new microservice-based software architecture; this version has not yet been deployed to our analysts since it is still under active development.

2 Background

Our research addresses software and UI design issues we have encountered when working in the field of crisis informatics and is related to the technical and non-technical challenges of storing, processing, and analyzing large amounts of social media data.

2.1 Crisis Informatics Research

Crisis informatics is an interdisciplinary research field that examines the socio-technical relationships among people, information, and technology during disaster and mass emergency events [2]. With multidisciplinary perspectives, researchers in this field study information flows—both offline and online—between members of the public, emergency management, and other important stakeholders via the use of communication and information technologies throughout the life cycle of a disaster event, including the preparation, warning, impact, response, and recovery phases [8, 13]. Technology in this sense, however, also includes non-computational means of producing and disseminating safety and time-critical information. With the widespread use of social media from geographically-dispersed locations, people deliver and discuss different types of disaster-related information. The expectations for this information—and the velocity, variety, veracity, and volume with which it is produced and shared—are now needed to gain situational awareness of the disaster event as it unfolds. This rich and in-depth information of social behaviors and interactions create raw data that is extensively tracked and interpreted by crisis informatics researchers.

2.2 Project EPIC

Our research group, called Project EPIC (Empowering the Public with Information in Crisis)—a large NSF-funded project (`http://epic.cs.colorado.edu`)—has been actively working in crisis informatics since 2007. Researchers in Project EPIC have been studying social media by members of the public and emergency management across hundreds of disaster events. Project EPIC defined a vision for technology mediated support in sharing disaster-related information to the public and emergency management personnel [2]. Project EPIC has enriched the crisis informatics field with research efforts to understand different types of social interactions and patterns that occur during these events. These efforts have quantitatively and qualitatively examined social media during mass emergency events to understand socio-behavioral phenomena of self-organization [14, 15], policy change [10], information sharing between unofficial and official sources [16, 9], and crowdwork [15, 17].

While a lot of research has been done in this area, research methods for collecting, storing, and making sense of these vast amounts of social media data are unwieldy, time-consuming, and expensive. The huge amounts of data generated from social media during these events create challenges in big data processing with respect to persistence, efficient retrieval, and analysis [6]. As software and web engineering researchers at Project EPIC, we seek to study how to design systems that create reliable and scalable infrastructure for big data processing and analysis, that support the analytical methods employed by research analysts in crisis informatics, and that embrace the ever-changing needs of analysts over time.

To solve these challenges, a data collection platform is required. Project EPIC's Twitter data collection system is called EPIC Collect [3, 5, 4]; it is a scalable, reliable, fault tolerant and 24/7 available system that is able to collect millions of tweets per day to capture the social media data generated for large-scale crisis events. Notice of an event, whether sudden (earthquake) or impending (hurricane, tornado), prompts our research analysts to identify relevant keywords for the event by monitoring the tweets that are being generated and by identifying place names (cities, regions, rivers, etc.) associated with the impacted area. Those keywords are then submitted to EPIC Collect via the EPIC Event Editor—a simple front-end web application—to initiate data collection. EPIC Collect detects the presence of this new event and submits its keywords (along with the keywords of all other active events) to Twitter's Streaming API. It collects tweets containing those keywords, indexes tweet JSON objects, and stores them in Cassandra, a scalable and reliable NoSQL data store.

After collection has ended, our focus turns to analysis. For that, Project EPIC analysts make use of EPIC Analyze, a system we developed to browse, filter, sample, annotate, and visualize large Twitter data sets. We have reported on the general software architecture of EPIC Analyze and its core functionality previously [6]. In this paper, we focus on the design issues that have to be solved in order to provide a useful and usable user interface to EPIC Analyze. We present EPIC Analyze in more detail in Section 5.

2.3 Social Media Data Challenges

Social media systems are important platforms for the public to communicate both locally and globally during mass emergency events. As a crisis event unfolds, social media platforms deliver large amounts of data about the event often within seconds of its start. However, storing, wrangling, and analyzing social media data is a significant software engineering challenge [4]. The vast amounts of social media data disseminated during crisis events contain citizen-generated content in addition to content generated by official sources of information. Unfortunately, most of this content is simply noise (ambiguous keywords, potential misinformation or disinformation, stale information, expressions of concern, etc.). Additionally, the techniques used to collect this data makes it difficult to decide if the sample is representative and if the sources are trustworthy and accurate [8, 13].

Analysis of social media data during disaster events is different from other domains. From a software engineering perspective, the vast amounts of heterogeneous data create challenges related to scalability, reliabilility, and efficiency [4, 1]. Most notably, the ephemerality of the data plays a critical role in designing technologies with high reliability and scale. Some social media services do not respond well to programmatic calls to retrieve historical data as the event unfolds, especially for high-frequency terms. In the 2015 Nepal Earthquake event, terms such as "earthquake", "nepal", "kathmandu", and "NepalEQ" matched millions of tweets in the first few days after the event. If your data collection system is not running and collecting at the start of the event, it can be very difficult to go back and retrieve the tweets that contain these terms at the very start of the event. For instance, Twitter's Search API will often provide just the last few hours of tweets containing high-frequency terms, which is problematic if, for instance, an event started when Project EPIC analysts are sleeping.

Furthermore, some of these terms are noisy (e.g., there are multiple earthquakes that happen each day) while others generate lots of duplicates that need to be winnowed to avoid overwhelming analysts attempting to make sense of the data as it streams in. Other challenges include dealing with changes in the metadata of tweet objects, dealing with spikes in activity, dealing with hardware failures (which will occur since we're collecting data 24/7 over the course of years; EPIC Collect has been running in some fashion since Fall 2009), and dealing with the need to filter datasets consisting of hundreds of millions of tweets down to representative sets of just thousands of tweets. Our previous work has tamed a lot of these challenges [3, 5, 4, 6, 1] but that work provides a hint as to how difficult it can be to even get to the point of designing usable user interfaces to systems that help analysts analyze and make sense of these large data sets.

3 Evaluation of EPIC Analyze's User Interface

EPIC Analyze has been in active design and development since Fall 2013. In that time, we have had multiple opportunities to respond to user feedback to make small improvements to the system. However, shortly after its initial deployment, we became interested in how well EPIC Analyze supports the analytical workflows employed by Project EPIC analysts to perform their crisis informatics research. We turned to task-oriented usability methods to evaluate whether these workflows can be performed with the EPIC Analyze user interface and whether it provides an enjoyable user experience [18]. We now describe a user interface evaluation that we performed via think-aloud sessions on a previous version of EPIC Analyze. Based on the results of these sessions, we developed the interface that EPIC Analyze now provides (discussed in Section 5).

3.1 The Research Analysts

We interviewed seven crisis informatics researchers who self-identify as information scientists with specializations that draw from other disciplines, such as geography, journalism, and computational social science. These analysts all work for Project EPIC and work on events ranging from the 2014 Carlton Complex Wildfires in Washington (40,000 tweets), the 2013 Boulder Flash Floods (1 million tweets), the 2013 Japan Earthquake event (1.9 million tweets), and the 2012 Hurricane Sandy event (22 million tweets).

Previous research examined the work processes and tools used by analysts in this domain [19]. In our own study, we observed that the analysts access data in diverse ways, from using commercial analytical tools, such as Tableau and Splunk, to developing their own programs using programming languages like R and Python. Furthermore, their work processes are different based on their research: one analyst reports that she uses social network analysis methods in Python to model relationships of Twitter users within a data set, but she is often concerned about the representativeness of the data set. To address this concern, this particular analyst wrote her own scripts to filter the data sets such that she has a representative sample that can help answer her research questions.

Some analysts have stronger programming skills and are comfortable with programmatically retrieving data from EPIC Collect, storing it in their own databases, and scaffolding web pages of their analysis on top of that data. Still others use basic—yet powerful—tools, such as spreadsheets. Related work [20] shows that there is considerable overlap in functional power and usability with these tools since they are so well-known and understood; this poses challenges for adoption of other tools, sometimes with greater analytical power, that also can support these workflows. However, these productivity tools do fall short in some use cases, such as annotating tweets with labels and comments, which is necessary for qualitative and quantitative analysis. To understand how EPIC Analyze, therefore, may become useful to these analysts, we performed think-aloud sessions to see if EPIC Analyze could solve some of their current struggles and support their current and future analysis workflows.

3.2 Think Aloud Protocol Runs

We performed a series of think-aloud protocol runs to gather feedback on the progress we had made on the system interface, which included full search of most tweet-based metadata, support for creating and searching annotations on tweets, and creating new datasets out of previously-run filters (see Fig. 1). Think-aloud runs are well-known as a traditional usability method that captures feedback on a task that the user performs without any help or guidance from the interviewer [21]. We prepared several tasks that analysts should be able to perform without any prior experience with EPIC Analyze. These tasks reflect the bulk of functionality that is available to use for any given data set in EPIC Analyze. In all seven interviews, we asked the research analyst to perform the following tasks or to answer the following questions:

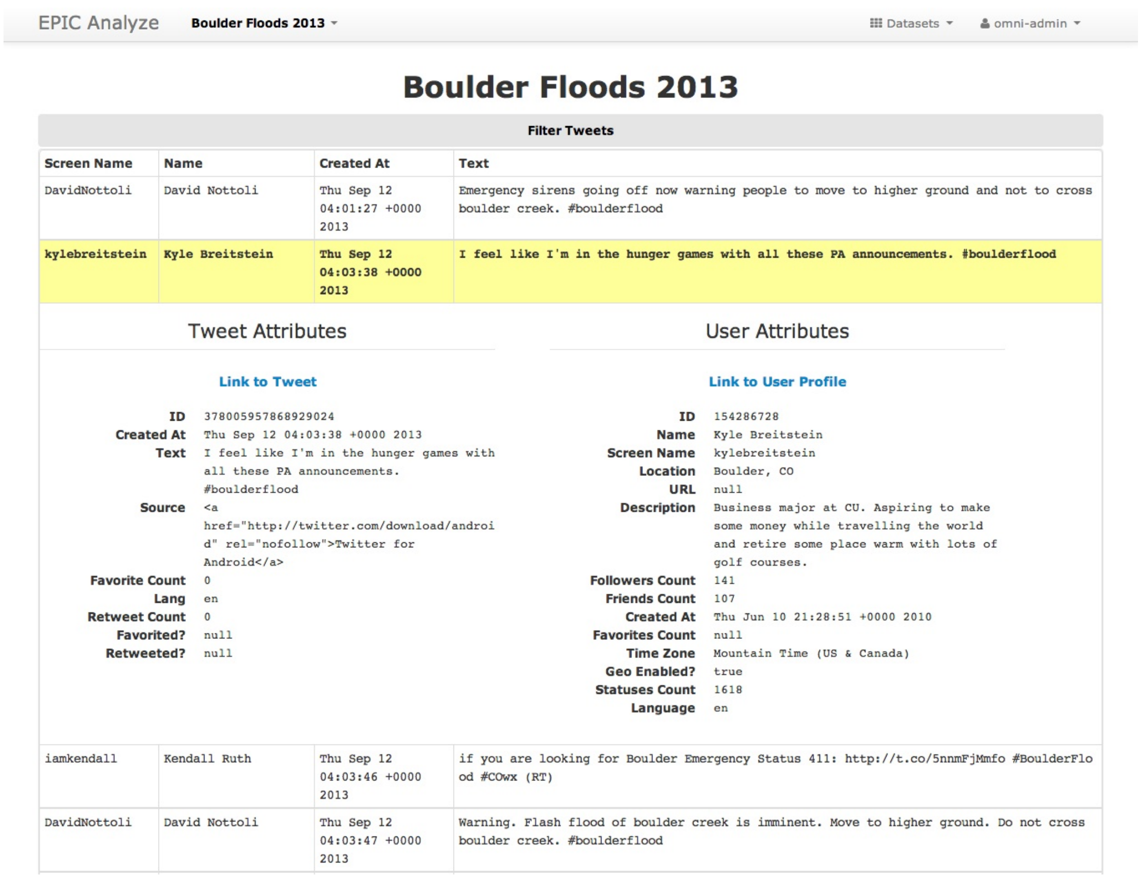
(i) Open an event that interests you.
(ii) Investigate a tweet and look at all of its attributes.
(iii) How can you view the tweets from only the first or last day of this dataset?
(iv) How can you get tweets written in Spanish? How about Spanish and French?
(v) Can you find tweets that have the word “bomb” or are from a specific user? Both?
(vi) How can you find tweets with annotations?
(vii) How can you comment on a tweet that has already been commented on?

Each research analyst completed the tasks without any difficulty. All of them enjoyed using the interface and the general look-and-feel of the application. About half of them found it surprising that certain filters—such as issuing a query for both Spanish and French tweets—was supported. Almost all analysts complained that there was no “reset” button at the bottom of the query interface to clear text and other selections from the form. Some analysts complained that the filter result notification bar—which displays the results of the filtered result set when the query has finished—should stay on the screen rather than fading away. This would create a reference point for the analyst in understanding how effective their query was. Also, they described the need for summative statistics, such as total dataset size, dataset date range, and keyword search terms (a feature already present but not very visible). One analyst suggested that all currently applied filters be displayed at the top of the search interface. All of this feedback was used to design the current version of EPIC Analyze (discussed in Section 5) and, indeed, all of these features are now present in that version.

After the tasks were performed, each subject was asked free-form questions about their experience with the interface. These semi-structured interviews at the end of the think-aloud sessions were critical to obtaining insight on the factors that would contribute to these analysts using EPIC Analyze in the future. We now describe additional issues raised by the analysts.

3.3 The Big Picture

Analysts described the desire to see a “big picture view” of the event. This term refers to the extent to which analysts understand the overall structure of an event under study and have access to the values of common metrics (e.g. the most retweeted tweet, the number of unique users, etc.) and useful visualizations (such as a histogram of tweet activity for an



Boulder Floods 2013

Screen Name	Name	Created At	Text
DavidNottoli	David Nottoli	Thu Sep 12 04:01:27 +0000 2013	Emergency sirens going off now warning people to move to higher ground and not to cross boulder creek. #boulderflood
kylebreitstein	**Kyle Breitstein**	**Thu Sep 12 04:03:38 +0000 2013**	**I feel like I'm in the hunger games with all these PA announcements. #boulderflood**

Tweet Attributes

Link to Tweet

ID	378005957868929024
Created At	Thu Sep 12 04:03:38 +0000 2013
Text	I feel like I'm in the hunger games with all these PA announcements. #boulderflood
Source	<a href="http://twitter.com/download/android" rel="nofollow">Twitter for Android</a>
Favorite Count	0
Lang	en
Retweet Count	0
Favorited?	null
Retweeted?	null

User Attributes

Link to User Profile

ID	154286728
Name	Kyle Breitstein
Screen Name	kylebreitstein
Location	Boulder, CO
URL	null
Description	Business major at CU. Aspiring to make some money while travelling the world and retire some place warm with lots of golf courses.
Followers Count	141
Friends Count	107
Created At	Thu Jun 10 21:28:51 +0000 2010
Favorites Count	null
Time Zone	Mountain Time (US & Canada)
Geo Enabled?	true
Statuses Count	1618
Language	en

iamkendall	Kendall Ruth	Thu Sep 12 04:03:46 +0000 2013	if you are looking for Boulder Emergency Status 411: http://t.co/5nnmFjMmfo #BoulderFlood #COwx (RT)
DavidNottoli	David Nottoli	Thu Sep 12 04:03:47 +0000 2013	Warning. Flash flood of boulder creek is imminent. Move to higher ground. Do not cross boulder creek. #boulderflood

Fig. 1. Previous Version of EPIC Analyze used in our Think-Aloud Sessions.

event). They admitted that viewing thousands of pages of tweets was daunting and that they would prefer digging into the data after queries returned a more manageable set of tweets to analyze, on the order of hundreds of tweets. In order to achieve that, they suggested, it would be beneficial to have a way of displaying volume of tweets over time. That way, they could identify areas of time that contained "spikes" in tweet activity that might be of interest to their research questions rather than having to solely rely on queries alone.

After the think-aloud sessions, development on this feature started immediately. As shown in Fig. 5, the current EPIC Analyze browser shows a timeline of tweet activity at the top of its display. This timeline is not just a static depiction of the volume of tweets for any given data set; it dynamically conveys volume of tweets for any query that is processed. This practice is in line with the recommendations made by Schneiderman and his Visual Information-Seeking Mantra: *Overview first, zoom and filters, then details-on-demand* [22].

3.4 Filtering Data Out

Our research analysts generally praised the ability to filter data based on conventional boolean operators such as AND and NOT, but they were also interested in filtering data *out* rather than qualifying data *in*. To do this, the unary operand NOT must be used to configure a query such that results that do not satisfy a filter are returned. Analysts claimed that this is useful because of the nature of their investigations. Sometimes the most meaningful data comes from official sources offering global insight [10] and sometimes from "unofficial sources" offering information local to an event [16]. As an example, an analyst may have a list of Twitter accounts that they know represents a list of official sources for information in an event. They can filter out tweets by these sources by first constructing a query with clauses that search for these accounts directly and then grouping those clauses with a NOT operand. Additional queries can then be combined with this one to look for specific information from "unofficial sources" (i.e. people local to an event).

Of course, implementing this extra operand poses challenges for the interface. It is easy to imagine placing checkboxes at the beginning of each filter on the query form representing whether the analyst would like to filter this input in or out. But that leads to a lot more choices that, if improperly designed, may pose confusion and additional cognitive load on the analyst. In the current version of EPIC Analyze, we have placed a drop-down list at the top of the filter form that includes three choices for filtering operators: AND, OR, and NOT. Analysts can choose exactly one option that will be applied to all filters in the interface at that time. Choosing the NOT operand will essentially group the filters with a locally implicit OR condition, and all filters are then prepended with a NOT clause in front of them. That is, a query with the clauses `(user.name = News9) AND (user.name = EmergencyPIO)` would instead become `(NOT user.name = News9) OR (NOT user.name = EmergencyPIO)`. This approach ensures tweets created by these accounts would not appear in the final results.

The reasoning behind this design decision is to abide by the observations we made during the think-aloud sessions and semi- structured interviews: analysts want to drill-down into data sets, not include as much data as possible. Specifically, when they are looking at a data set with millions of tweets, filtering out data is a primary and important action that allows them to dig deeper into the data set for specific signals. Providing the analyst the ability to include queries made up of NOT and OR operators over just the use of the AND operator achieves this: they are able to select more specifically what data will be filtered out. In future user studies, we plan to evaluate whether this mode of filtering out is preferable over using exclusive joins (AND clauses) between the filters.

The user interface task is not done after implementing the NOT operand, however. What if the analyst wants to make compound queries using these operators, such as "(X OR Y) AND Z"? How can the interface and the system support this? To solve this problem, we had to make use of more advanced techniques that included modifying the data model of EPIC Analyze as a result. We present details on these more advanced techniques in Section 4.

4 Query Support

We report here on the architectural design of the query facilities implemented within EPIC Analyze. Querying is an integral component of a data analytics infrastructure. If querying is not supported, users of the analytics infrastructure will not be able to retrieve data efficiently. We performed the work discussed in this section to respond to the feedback received from our analysts in Section 3. Without this new approach to handling queries in EPIC Analyze, it would not have been possible to support the new user interface discussed in Section 5.

We turned to data modeling techniques to address querying as a first-class concern in EPIC Analyze. Data modeling in data-intensive systems is a key technique in making them reliable, scalable, and efficient [1, 3, 4, 5, 6]. With respect to EPIC Analyze and its support for searching and filtering large data sets, we have observed that the data modeling decisions made in the lower layers of the software architecture impact how well EPIC Analyze is able to respond to query requests. In this section, we describe the server-side querying facilities that respond to the queries made by an analyst. We illustrate this architecture by stepping through the typical request-response cycle that is generated in response to a query; we then discuss two data modeling challenges we faced in providing a good user experience and the techniques we employed to solve them. We connect these lessons with the impact that they had on improving EPIC Analyze's user interface.

4.1 Query Modeling

The query architecture of EPIC Analyze (see Fig. 2) is composed of various components to properly serve HTTP requests coming from the user interface. Some of these components are tied to the application framework (Ruby on Rails), such as the Datasets Controller, the QueryDispatcher, and the Model interfaces, and some of them are acting as independent entities. For example, the EPIC Gem—a Ruby gem that helps query data sources like Cassandra, Solr, and Redis—is a separate library that is integrated into the EPIC Analyze web application while also serving other external use cases. We present a typical request-response cycle to properly serve a query from the web interface and explain how each component serves an important function in this process.

The following models and components are used to support queries in EPIC Analyze:

- **Query**: A model class to encapsulate a query string received from the web interface.
- **QueryChain**: A model class designed to manage, iterate, filter, and order Query objects. There is a one-to-many relationship between QueryChain and Query.
- **Dataset**: A model class for defining a set of tweets filtered by Query instances. A

Dataset instance relies upon its sole QueryChain instance to manage Queries that represent its filters.

- **QueryDispatcher**: A component that transforms an HTTP request for filtering tweets into a Query object and then executes it based upon rules and available data sources.
- **Datasets Controller**: A Ruby-on-Rails component that routes a HTTP URI to an action. When tweet filters are populated and then submitted by the user, the Datasets Controller receives this payload and calls the QueryDispatcher to properly execute the request and return the response back to the user.
- **Annotation**: A model class to store user-based comments and labels in Postgres. Query instances may be subclassed as AnnotationQuery instances if the HTTP request contains annotation-based filters.

The model classes are defined as classes in ActiveRecord, the object-relational mapping framework for Ruby on Rails. Additionally, it is important to highlight that the primary social media data being operated on by this query infrastructure is tweets, but this infrastructure can operate on other social media objects with just a few modifications. The common denominator between and among the data sources used in our analytics infrastructure—Solr, Hadoop, and Redis (see Section 5)—are the tweet id and the row key that is used to store a tweet in Cassandra. Each data source may store additional information, such as enrichments on top of the original Twitter data, but they are always bound to a tweet id and row key that are linked back to the primary authoritative data source, Cassandra. We refer to pairwise units of tweet id and row key as *tweet references*, because they uniquely identify a tweet in our Cassandra data model. We use tweet references in all data sources so that a tweet's data can be retrieved from Cassandra. More information on the data modeling decisions in our persistence layer with respect to these tweet references can be found in [5, 6].

4.2 A Day in the Life of a Query

A typical search request for annotations or tweets in EPIC Analyze is made on the web interface and is illustrated by the following request-response cycle:

(i) A request to issue a query through the web interface begins with a set of filter parameters that are sent via a HTTP GET request. The request is received by the Datasets Controller action named `tweet_search`, which is responsible for unpacking the request, loading the correct Dataset instance (which represents the current set of tweets), and invoking the QueryDispatcher to properly route the request's filter parameters to the appropriate data sources.

(ii) The QueryDispatcher receives the request payload and the current QueryChain object managed within the current browser session. The filter payload is sent through the Query constructor and—based on its parameters—is transformed into a Query subclass, either AnnotationQuery or SolrQuery. The main objective of QueryDispatcher is to send the QueryChain through to neighboring functions that process the query against our data stores (Postgres and Solr). In this case, an AnnotationQuery was created, and so QueryDispatcher passes the QueryChain instance to the annotation `search` function.

(iii) The annotation `search` function searches the QueryChain instance for AnnotationQuery instances, passes them to the Annotation model, and returns search results of matching

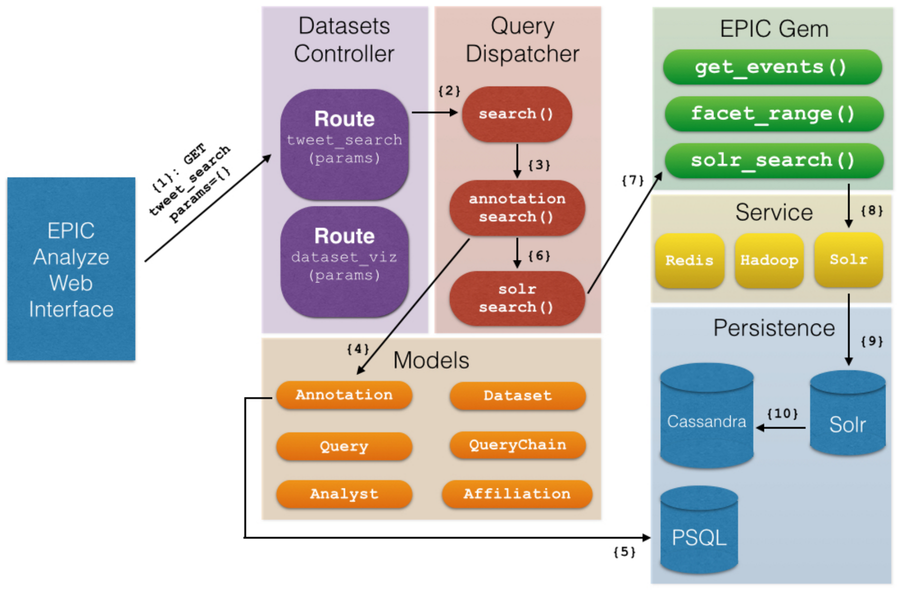


Fig. 2. Query Life Cycle.

tweet references. This function is designed to aggregate the results from the AnnotationQuery instances found in the QueryChain and honor the logical operators that were expressed for each query. Since the system imposes a global AND operation between all Query instances, if an AnnotationQuery returns no results, we can short circuit the entire search operation and return no results back to QueryDispatcher. It will, in turn, return no results back to the browser.

(iv) The Annotation model interface (i.e. the class generated for Annotations by ActiveRecord) offers functionality for searching Annotation instances based on tweet ID, label name, label type, or comment. In this case, its search function is called with the current AnnotationQuery parameters and searches for Annotation instances that satisfy them.

(v) Rails provides object-relational mapping facilities for linking model interfaces with various database systems, in this case Postgres. Annotation instances are returned from calls to the Rails models and control flow is returned back to the QueryDispatcher.

(vi) At this point within the QueryDispatcher, either there are tweet references that have been returned from the Annotation model or not. If no results are returned, then logically-speaking, no further querying can satisfy the current request, so the QueryDispatcher will return nothing back to the datasets controller. In the normal case, however, control flow will continue with annotation-derived tweet references being sent to the Solr search function.

(vii) The Solr `search` function collates SolrQuery instances from the QueryChain object and uses the EPIC gem to construct a Solr-based query. The function combines the query strings from these SolrQuery instances with any annotation-derived tweet references so that the impending Solr query is further constrained by the ids of those tweets.

(viii) The corresponding custom search call is made within the EPIC gem; it accepts the query from the QueryDispatcher and makes the call to Solr.

(ix) Solr executes the query according to the parameters and returns paginated tweet references back to the QueryDispatcher.

(x) With a paginated payload of tweet references, the QueryDispatcher must now resolve the tweet references to retrieve the full tweet JSON object from Cassandra. It achieves this by calling the EPIC gem to make a batch call to fetch page-sized JSON objects from Cassandra. Once the resolved tweets are returned, the QueryDispatcher wraps them within Tweet object instances and returns them to the datasets controller, whereupon they are formatted into a response payload that is returned back to the analyst.

Despite the complexity of this workflow, most queries are resolved within a few seconds. Queries that have been created and served by this process are saved in memory within the QueryChain object, which is passed from one request to another via the session. Note that the QueryChain object is not persisted to Postgres until the analyst decides that the string of queries that have been issued is desirable to create a new Dataset object for more localized analysis. In fact, any analyst may choose to save a new Dataset instance after issuing any number of queries, which will populate the underlying QueryChain object. To properly serve the new Dataset instance, EPIC Analyze retrieves its corresponding QueryChain, executes the queries within it, and accepts any incoming requests to append to the chain.

This request cycle is the culmination of work done to solve challenges that were identified by our analysts during our usability tests discussed in Section 3. We now describe these challenges followed by the approach we used to solve them.

4.3 Query Visibility

While interfaces from tools like Google Refine, Wrangler, and Splunk have powerful ways to query against large amounts of data, some of them struggle with providing visibility with respect to their effectiveness. Google Refine will display updated counts on facets and the total number of hits, but that does not describe when or where the query affected the data most. Splunk does have a visual timeline that updates itself after each query but falls short of providing usable facilities to perform previous queries, or to string them together without having to learn a query language based on pipe commands. Not enough of these tools promote the tangible aspects of the queries that are issued: how to create, jump between, and destroy them, as well as to view their history. That is why we implemented the interface described in Section 5; it *directly* supports these features. To make this possible, we had to implement two new domain models: Query and QueryChain.

In previous versions of EPIC Analyze, querying was relatively straightforward. A simple filter form was designed to accommodate filtering most fields, including the text of a tweet, retweet count, date range, the presence of URLs, the presence of geo-coordinates, and more. After the user submitted the filter, the set of filter parameters was transformed into parameters that could be used to submit to Solr. This achieved basic querying against large data sets, but it did not provide any visibility into the queries after the result set was displayed. Recall that most of the research analysts we interviewed felt as though the interface was missing the "big picture" of the event. They did not want to be scouring through thousands of tweets, page after page. They wanted to see how their queries were making a difference in filtering out the data.

These issues of visibility and tangibility were solved with the Query and QueryChain objects. As mentioned before, the Query object is an encapsulation of data attributes coming from the filter parameters on the web interface, and the QueryChain object keeps track of the Query instances created and groups them by instance type to run queries against the appropriate data source. With these querying facilities established, it becomes straightforward to display them in the interface. A "Current Filters" section in the filter form displays a list of all of the queries that have previously been made. Analysts now are able to create new queries that show up on the list after they have been processed. Analysts can also click on any one of these queries, and the Datasets Controller will identify an index that was linked to the selected query. The QueryDispatcher is then able to rearrange a subset of the queries with respect to the selected Query instance and execute them. The results reflect the selected query, as well as the queries that came before it in the QueryChain. We refer to this user-action as query-jumping because the analyst is able to jump between previous queries to view the results that each one has made in drilling down into the data set. This provides visibility with respect to the impact of each query, as well as gives control to the analyst in how to redirect analyst-made queries if, e.g., the most recent query was not effective.

Additionally, analysts can delete any query along the chain, as shown in the filter form (see Fig. 5) with the X-mark next to each query name. Once deleted, the query will be removed from the QueryChain object, and the analyst will be able to continue making queries from the latest point while ignoring the effects of the deleted query. These query-based controls empower the analyst to make decisions without consequence: that is, the basic yet crucial operations that can be performed on queries is the basis for how analysts can drill down into the data set to identify the most important features for their research questions.

4.4 Query Expressivity

The ability to perform these operations on queries is critical for exploratory data analysis on large data sets, but not if the expressiveness of queries is not powerful enough. Based on our think-aloud sessions, we learned that the most basic querying operators—AND, OR, and NOT—reflected the kinds of queries that analysts wanted to make, such as: "Give me all of the tweets in the 2012 Hurricane Sandy data set that are geotagged, and then take away all the ones that have the screen name *spammer123*." This kind of natural inquiry should be possible, assuming that the relevant metadata fields are indexed by, in this case, Solr. However, what if the analysts want to make an additional query, such as, "Now, remove tweets that have been tagged with *social media* or *local/state government*." How can a web interface provide the flexibility and expressivity for querying against multiple data sources? Current analytical tools either do not support this or struggle to provide timely responses for both dataset-specific and user-generated data.

Unbeknownst to the analyst, the nature of this problem requires insight into the complexity of algorithms needed to satisfy such heterogeneous queries. We observe here that there are naive ways to perform them, such as a brute force result set intersection: let the Query-

Dispatcher have the annotation search return tweet references based on the AnnotationQuery instances, and then let Solr return tweet references based on SolrQuery instances. The resulting work now involves having to intersect these two result sets. However, the problem is that the result set coming from Solr is arbitrarily large—Solr indexes datasets containing millions of tweets. Performing the intersection between annotations and tweets in this instance would be burdensome and take too long to return back to the user at interactive speeds.

Again, data modeling helped us find a solution. Recall the search request-response cycle in Fig. 2. The Annotations database will return result sets that are sufficiently smaller on average than those returned by Solr. This is evident because while EPIC Analyze supports annotations on tweets during exploratory analysis, most analysts will not manually annotate more than a few thousand tweets. Therefore, annotation-based tweet references can be processed independently before searching Solr, regardless of the order of Query instances in the QueryChain.

Indeed, the QueryDispatcher will send the QueryChain object to its annotation search function to collect the tweet references returned by all of its AnnotationQuery instances. For each query—issued with the AND, OR, or NOT operator by the analyst—the results are stored in a data structure we refer to as tweet reference groupings; each group contains tweet references and the chosen operator for that query. Once complete, the annotation-based tweet reference groupings are fed into Solr as an extra argument, which strings together the tweet ids and joins them with the logical AND operator. That way, local filter operators and the implicit global AND between all queries in the QueryChain are represented, and Solr will process a result set that is constrained by the tweet reference groupings generated by the annotation search.

This approach works solely because tweet references are the lowest common denominator among these data sources. If we were to include additional sources that provide further enrichments to the user experience, the data modeling would then easily extend to those technologies if they also persisted tweet references. For our application infrastructure, we have seen no observable latency in Solr calls given that annotation-based tweet reference groupings are part of the Solr query. Indeed, since Solr is deployed on a powerful set of machines in our computing environment, query results are usually completed in a few seconds, even for datasets consisting of millions of tweets.

5 EPIC Analyze

As discussed above, our software engineering team has been iteratively developing EPIC Analyze since 2013 to allow Project EPIC analysts to browse, filter, share, and visualize large-scale data for crisis informatics research. In this section, we present its heterogeneous architecture which integrates multiple technologies, services, and tools to provide an analytics environment for performing data analysis at scale. We then give a broad overview of the EPIC Analyze system and its features that are the result of the work we performed in response to the user study discussed in Section 3

5.1 The EPIC Architecture

The architecture for EPIC Analyze (see Fig. 3) builds on top of the software architecture of EPIC Collect [3, 5]. EPIC Collect's architecture consists of the EPIC Collect service which connects to Twitter and retrieves tweets based on event information that is managed by the EPIC Event editor. The event information maintained by that web application is stored in Cassandra, a NoSQL columnar data store, that also stores all of the tweets that stream in from Twitter's Streaming API. We have written extensively about the data model that we use within Cassandra (building on top of Cassandra's own data model of keyspaces, column families, rows, and columns) to scalably and reliably store tweets within Cassandra [6, 23].

EPIC Analyze is then layered on top of Cassandra using a variety of additional technologies to achieve various scalability, reliability, and efficiency goals. Postgres is used to store the annotations that analysts make while studying a large Twitter data set. Solr is used to provide full-text search capabilities over the tweets stored in Cassandra. Pig and Hadoop are used to perform batch processing tasks on the tweets with the results of those jobs being stored back in Cassandra in column families separate from the ones that are used to store the original collected tweets. Redis is used to help accelerate various operations by, for instance, caching the results of queries or indexes that are used to sort tweets in memory.

EPIC Analyze is a Ruby on Rails web application that efficiently makes use of the software frameworks and technologies provided by the underlying layers to allow analysts to view and manipulate the collected datasets. This architecture is extensible which allows for the integration of third- party tools, such as Splunk, which then simply adds additional features that our analysts can use while performing their analysis tasks. With this architecture, a Project EPIC analyst can efficiently study Twitter data sets consisting of millions of tweets. Having discussed this architecture at a high-level, we now go into more detail concerning some aspects of this architecture and EPIC Analyze's functionality below.

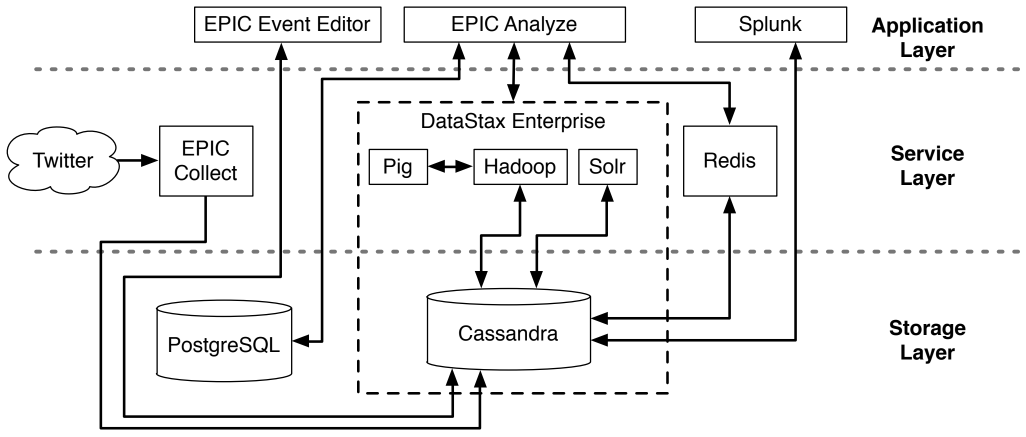


Fig. 3. EPIC Analyze Architecture.

The service layer of the EPIC Analyze software architecture hosts technologies that are used for collecting, caching, indexing, and otherwise processing data. Beyond EPIC Collect, services such as Hadoop, Solr, and Cassandra are tightly integrated into a data processing framework known as DataStax Enterprise (`http://www.datastax.com/`) and are used to filter and process the data served to the user by EPIC Analyze. DataStax Enterprise contains a version of Hadoop, Solr, and Pig that have been modified to integrate with Cassandra and these technologies therefore know how to read and write the Twitter data sets stored in Cassandra column families. Pig provides a high-level scripting language—Pig Latin[24]—that auto-generates Hadoop jobs that then access the Twitter data stored in Cassandra.

Solr provides full-text search and filter capabilities on the tweets stored in Cassandra. However, we cannot apply those features directly on the data stored in EPIC Collect because they are stored in the column families using their "raw" format: JSON. Therefore, we need to migrate the raw JSON data into a new column family in Cassandra that normalizes each tweet to a standard set of fields that then act as a schema that Solr can index. EPIC Analyze also uses Redis—a key-value in-memory store—as a caching mechanism to achieve excellent performance when processing frequently accessed data.

In the storage layer, we make use of both NoSQL (Cassandra) and RDBMS (Postgres) database systems. For performance, Cassandra is deployed across a four-node compute/storage cluster. Postgres runs on a single machine but the amount of data it needs to store with respect to analyst-created annotations is relatively small and therefore a single machine is sufficient. Additional information about the software architecture of EPIC Analyze can be found in our previous work [6].

5.2 The EPIC Analyze Application

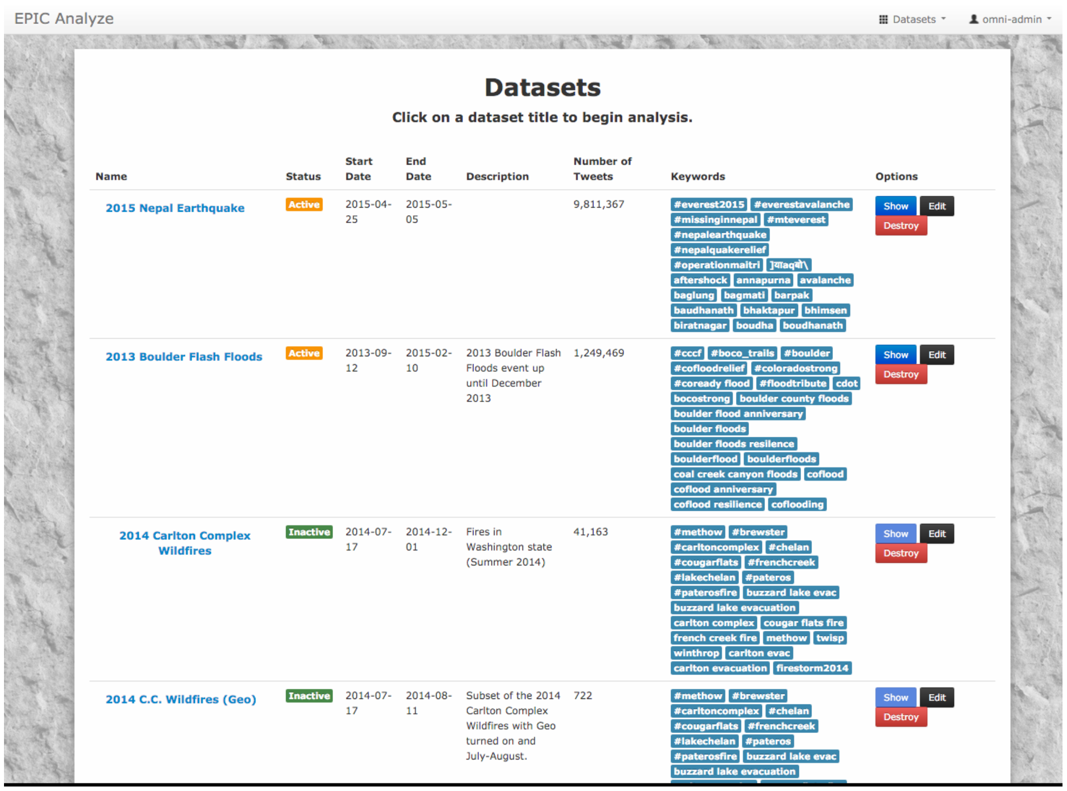
As mentioned, EPIC Analyze is a web application that provides scalable and efficient filtering, analysis, and annotation capabilities on large Twitter data sets. We now provide additional information on these features.

5.2.1 Browsing, Searching, and Visualizing

When an analyst logs in, she sees a list of data sets (see Fig. 4) that have been indexed by EPIC Analyze (e.g. "2015 Nepal Earthquake"). Once a data set has been selected, an analyst can view the tweets page-by-page in the EPIC Analyze browser (see Fig. 5.c). The browser provides an overview of the data set via a timeline that shows the volume of tweets over time at the top of the browser. On the right hand side, a detailed view of a single "page" of fifty tweets is displayed. On the left hand side, a form for querying the data set and its annotations is presented. If an analyst clicks a tweet, all of its relevant metadata is displayed in an in-line form for easy viewing; this form also contains links that take the analyst to see the original tweet on Twitter. On the timeline, analysts can click and drag (see Fig. 5.a) to specify a start and end date that will be used for all subsequent queries.

The filter form on the left allows analysts to search the dataset by tweet or by annotation. The tab for tweet-based search presents a list of tweet attributes that can be used to filter an entire dataset. The form supports standard boolean operations for advanced search; any number of tweet attributes can be used to specify a query. Back-end services, such as Solr and Postgres, are used to implement these queries and to provide facets. For instance, an analyst can click the *keyword* filter field to see a drop-down list of all keywords associated with that dataset (along with the number of times each keyword appears in the dataset). This provides analysts with an idea of how popular (or unpopular) a certain keyword was in that data collection and may guide or refine the questions they ask of the dataset.

The tab for annotation search offers the analyst the ability to make queries against user-generated content. Annotations include both labels and comments and are visible to all



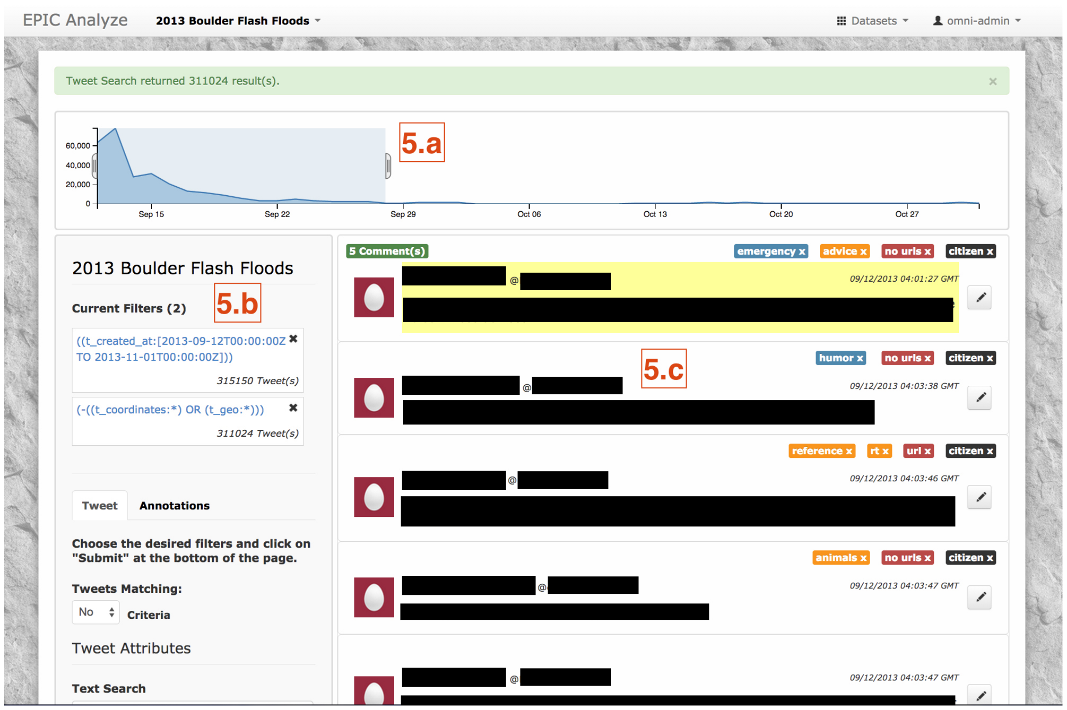
Datasets

Click on a dataset title to begin analysis.

Name	Status	Start Date	End Date	Description	Number of Tweets	Keywords	Options
2015 Nepal Earthquake	Active	2015-04-25	2015-05-05		9,811,367	#everest2015 #everestavalanche #missinginnepal #mteverest #nepalearthquake #nepalquakerelief #operationmaitri]याaqयो\ aftershock annapurna avalanche baglung bagmati barpak baudhanath bhaktapur bhimsen biratnagar boudha boudhanath	Show Edit Destroy
2013 Boulder Flash Floods	Active	2013-09-12	2015-02-10	2013 Boulder Flash Floods event up until December 2013	1,249,469	#cccf #boco_trails #boulder #cofloodrelief #coloradostrong #coready flood #floodtribute cdot bocostrong boulder county floods boulder flood anniversary boulder floods boulder floods resilence boulderflood boulderfloods coal creek canyon floods coflood coflood anniversary coflood resilience coflooding	Show Edit Destroy
2014 Carlton Complex Wildfires	Inactive	2014-07-17	2014-12-01	Fires in Washington state (Summer 2014)	41,163	#methow #brewster #carltoncomplex #chelan #cougarflats #frenchcreek #lakechelan #pateros #paterosfire buzzard lake evac buzzard lake evacuation carlton complex cougar flats fire french creek fire methow twisp winthrop carlton evac carlton evacuation firestorm2014	Show Edit Destroy
2014 C.C. Wildfires (Geo)	Inactive	2014-07-17	2014-08-11	Subset of the 2014 Carlton Complex Wildfires with Geo turned on and July-August.	722	#methow #brewster #carltoncomplex #chelan #cougarflats #frenchcreek #lakechelan #pateros #paterosfire buzzard lake evac buzzard lake evacuation	Show Edit Destroy

Fig. 4. The Datasets View of EPIC Analyze.

analysts working on a data set to foster collaboration during the analysis process. These annotations can be queried using the same logical operations and faceting capabilities described above for tweet-based search. Submitted queries are processed quickly, often within a few seconds; we credit this performance to the design of EPIC Analyze's software infrastructure.



Tweet Search returned 311024 result(s).

2013 Boulder Flash Floods

Current Filters (2)

((t_created_at:[2013-09-12T00:00:00Z TO 2013-11-01T00:00:00Z]))
315150 Tweet(s)

(-((t_coordinates:*) OR (t_geo:*)))
311024 Tweet(s)

Choose the desired filters and click on "Submit" at the bottom of the page.

Tweets Matching: No Criteria

5 Comment(s) — emergency x, advice x, no urls x, citizen x
[redacted] @ [redacted] 09/12/2013 04:01:27 GMT
[redacted]

humor x, no urls x, citizen x
[redacted] @ [redacted] 09/12/2013 04:03:38 GMT
[redacted]

reference x, rt x, url x, citizen x
[redacted] @ [redacted] 09/12/2013 04:03:46 GMT
[redacted]

animals x, no urls x, citizen x
[redacted] @ [redacted] 09/12/2013 04:03:47 GMT
[redacted]

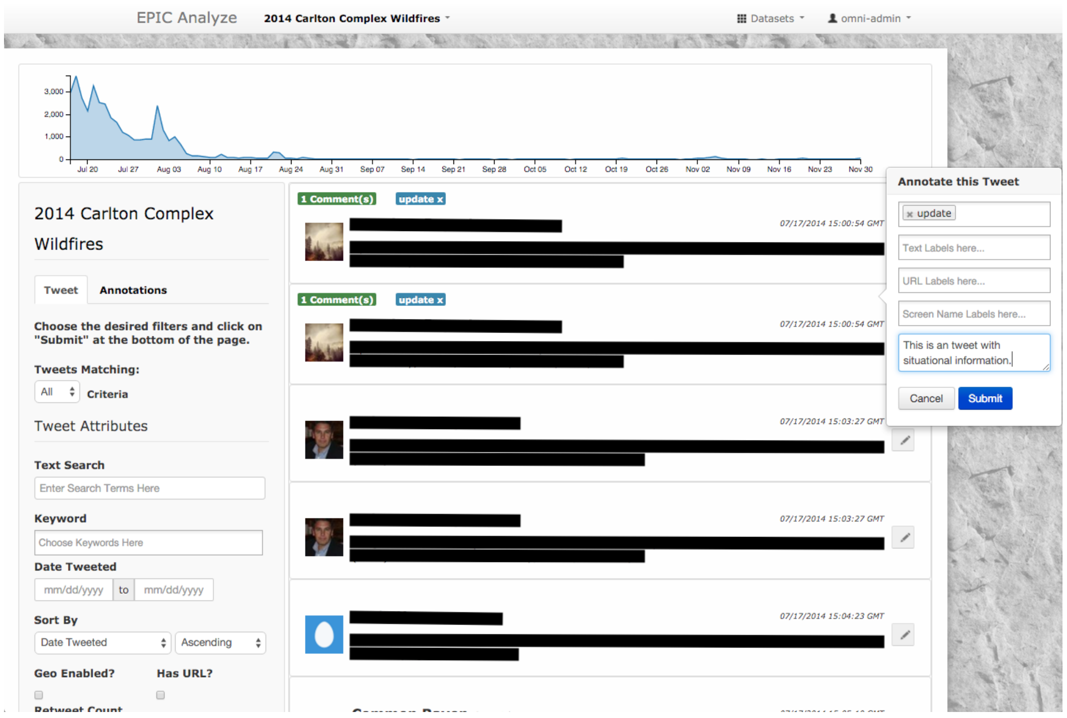
[redacted] @ [redacted] 09/12/2013 04:03:47 GMT
[redacted]

Fig. 5. The current version of EPIC Analyze.

After a query has been submitted and the result set has been filtered to match, the query appears in the "Current Filters" section of the user interface (see Fig. 5.b). This list provides a summary of the queries that were performed in the past and that are in effect as the analyst drills down further into a dataset. Analysts can choose to delete a filter in this list or they can jump to a filter by clicking on its name. Such a click will render the result set for the filters up to that point. This first-class interface for queries provides a more tangible experience when analyzing data sets, and places the analyst in control over the analysis process.

5.2.2 *Annotating Tweets*

If an analyst wants to annotate a tweet, she can click on the pencil icon that appears on the right side of each tweet; this action causes an annotation form to appear next to the tweet (see Fig. 6). The analyst has the option to annotate the entire tweet or to annotate just portions of the text of the tweet. In the latter case, the browser updates the annotated text to appear in a color associated with the label, making the annotation readily identifiable in future analysis sessions. Analysts can also comment on the tweet; multiple comments appear in a conversation thread (one per tweet) that appears in the annotation form. The browser indicates that comments exist for a particular tweet by displaying the number of comments for a tweet at the top of its display (see Fig.5.c).



Annotate this Tweet: update

This is an tweet with situational information.

[redacted]

Fig. 6. The Annotation Editor of EPIC Analyze.

The annotation of tweets is a recognized and critical activity for crisis informatics research. Methods for doing so are documented in empirical studies [10, 9, 16, 15]. However, before our work with EPIC Analyze, the process of annotating tweets in crisis data sets was laborious and error prone, tolerated only because it was the only way to conduct the research at the time. EPIC Analyze's support for annotations has been welcomed by Project EPIC analysts; they especially like its ability to allow search and filtering over the annotations. However, future enhancements have been identified and include automating the tagging of tweets with labels—especially for large data sets—and allowing analysts to create and/or load their own labels (i.e. coding schemes). A label is often more than just a textual annotation but is instead a mapping between values found in a tweet and a particular label. For example, analysts may want to tag all existing and future tweets that come from the user `News6` as "local media." We intend to add these features to EPIC Analyze in the future.

5.2.3 Other Features

Research analysts can perform other useful functions with EPIC Analyze. For example, once they have applied a set of filters against a data set, and the resulting set of tweets is useful for future analysis, they can save the state of the result set as a new data set. With this feature, analysts can now engage in more localized analysis with other analysts without having to search a large data set from scratch each time. Secondly, as previously documented in [19], the ability to export these data sets in well- known formats is critical. For this reason, EPIC Analyze allows analysts to export result sets to CSV. Additionally, all annotation labels and comments that have been used to tag the tweets that appear in the result set also appear in the CSV. Finally, with the help of Redis and a job framework known as Resque, EPIC Analyze provides analysts and administrators with the ability to write Hadoop jobs or other scripts that process each tweet in a data set. This job framework is already used by EPIC Analyze to automatically sort large Twitter data sets by multiple sort dimensions, such as tweet id, screen name, and retweet count [23]. The job framework allows many new features to be integrated into EPIC Analyze; it is one of the primary extensibility mechanisms that EPIC Analyze offers to developers [6].

6 Migrating to Microservices

Data intensive software systems, such as EPIC Analyze, must solve technical challenges of scalability, responsiveness, and reliability so that research analysts can use them to analyze datasets consisting of millions of elements. When developing the version of EPIC Analyze discussed in Section 5, we were struck with the high coupling that existed between the components that implement our query engine. The Query and QueryChain model classes, the Datasets Controller and QueryDispatcher components, and the EPIC Gem library were all implemented with the goal to enable fast, ad-hoc exploration of the social media datasets collected by EPIC Collect. To add new features—such as the ability to query against a real-time data stream or to expand EPIC Analyze to handle other types of social media—would require significant re-work and re-design within the application, both with respect to EPIC Analyze's user interface as well as to its server-side components.

We also realized that the data we collected in Section 3 revealed that not all of our analysts want to use the user interface provided by EPIC Analyze. Instead, they prefer to export a large data file straight from EPIC Collect, write their own script to convert it to a format needed by a third-party analysis tool, and then work with the data in that application. Others are fine with using EPIC Analyze's web-based user interface but need more flexibility with respect to the types of data it operates on and the types of data it exports. Since our top priority is to meet the needs of our analysts, we need to identify ways in which EPIC Analyze can support—not alter—their workflows.

To respond to these concerns, we decided to migrate EPIC Analyze to a software architecture that makes use of microservices [25]. The goal was to loosen the coupling of the server-side components, increase their individual cohesion, and allow for the addition of new functionality via the deployment of new services. We now present the details of this migration which represents a significant new contribution on top of the work we presented in the original paper [12]. We begin by discussing the tradeoffs we made in switching to a microservices approach. We then describe the new architecture that EPIC Analyze now has as a result of this migration. This work allowed us to reflect on limitations present in the monolithic version of EPIC Analyze and we conclude this section with a discussion on how the use of microservices allowed us to address those limitations.

6.1 Microservices Architecture

With the era of big data, quickly evolving requirements, and the need for continuous delivery, software development techniques and software architectural styles have been moving away from building monolithic applications in favor of a more modular approach. The more modular approach to software design is one that breaks applications into a set of highly cohesive and loosely coupled software entities that all operate independently. Despite this independence, they can be combined at run-time into an application with an extensible set of functionality and the elastic capacity to respond to a wide range of scaling scenarios.

This migration to modular software design has been happening for a long time starting with the rise of object-oriented design principles and object-oriented programming languages [26], then component-based software engineering [27] and RESTful web services [28]; followed, most recently, by service-oriented computing and service-oriented architecture [29]. Now with the rise of data-intensive software systems, there is a need for even more modularity and flexibility. Data- intensive software systems are often assembled out of much larger software frameworks that are each designed to be scalable in some way.

The software architecture for Project EPIC (see Fig. 3) is an example: Cassandra is a data store that can run on a cluster of machines; Postgres is a time-tested relational database; Hadoop, Solr, and Redis are all designed to run on clusters of machines, leaving only a small number of components that need to be implemented by hand. Marz and Warren [30] describe the desired characteristics of data-intensive software systems: robustness, fault tolerance, low latency reads/updates, scalability, generalizability, extensibility, ad hoc queries, and minimal maintenance. To provide these properties, developers need a flexible development approach that provides the freedom to respond rapidly to changes in technology and requirements.

Microservices represent a new approach for designing and developing software applications that builds on the legacy of service-oriented architecture [31], while avoiding some of the complexity of that approach's initial implementations (e.g., WS-Choreography, BPEL, and the heavy-weight enterprise service bus [25]). The basic idea behind microservices is that developers take a major piece of functionality in a system and decompose it into a small set of services that are developed, tested, deployed, and maintained independently, and communicate with each other using standard techniques from REST-based web services. The use of REST and HTTP-based APIs increases the *interoperability* of the overall system since documenting and using REST-based web services is well understood. This interoperability is then critical to the success of data-intensive software systems, allowing them to be assembled from much smaller parts. Accordingly, the word "micro" in microservices strongly suggests that each service should perform a minimal set of tasks—ideally one—that serves to decouple the service from the rest of the system [25].

In the context of EPIC Analyze, our decision to switch from a monolithic web application to a web application based on microservices is primarily due to the potential microservices have to make it easier for EPIC Analyze to be extensible, independent, and accessible.

In moving EPIC Analyze to a software architecture based on microservices, we make it possible to readily embody these characteristics into the overall system. Extensibility speaks to the capacity to build an environment with discrete building blocks; each one serving a specific function but together forming a suite of functionality for social media data analysis. Independence enables separation of concerns for both developers and analysts alike allowing the use of different languages, tools, and data sources to construct services useful for any client within the environment. Finally, accessibility means that we place value in the environment's ability to spin up new access points that are distinct from the HTML-based interface for the web browser and allow users to access specific services or export data in specific formats and thereby increase the likelihood that analysts make use of EPIC Analyze.

6.2 Project EPIC Microservices

Using a microservices-based approach, we have redesigned EPIC Analyze such that it can solve the problems mentioned at the start of this section and achieve the characteristics of extensibility, accessibility, and independence discussed above. We have refactored the server-side components for EPIC Analyze and wrapped them in services using the easy-to-adopt Sinatra web framework (`http://www.sinatrarb.com/`). The new services that now support the next version of EPIC Analyze are:

- **QueryDispatcher**: Like the QueryDispatcher component, the QueryDispatcher service provides a unified interface to query against multiple data sources. In its service form, it is responsible for encapsulating query components that know how to access particular data stores. These components have access to the services below and will call on them as needed to retrieve the results of a submitted query. This service functions as an API gateway to the rest of Project EPIC's new services; direct calls to these services can always be made, but calls via the QueryDispatcher take advantage of its knowledge of how best to orchestrate the work of the other services.
- **Authentication Service**: This service manages the meta-data associated with analysts, not unlike the Metadata-as-a-Service model presented by [32]. The Authentication Service persists and authenticates queries submitted by analysts before those queries are permitted to call the other services in the environment. This protocol is put in place to safeguard Project EPIC data against unauthorized use. When an analyst is added to the system, an administrator assigns an affiliation to the analyst's profile. The affiliation object then determines what access rights the analyst has with respect to the data sets contained in EPIC Analyze and EPIC Collect. We added these features to ensure that third-party collaborators could view Project EPIC data but only data related to their specific collaboration. This functionality is required due to Twitter's terms of service; we are only allowed to share the Twitter data that we collect with people who are direct collaborators on the project that caused a particular dataset to be collected.
- **Dataset Service**: This service provides an API for operations on Dataset, Query, and QueryChain objects. With this service available, analysts can save their work in the form of new Dataset instances and share it with other analysts. Note this functionality was available in the monolithic version of EPIC Analyze, but now any client application can make use of this service to do the same thing.
- **Tweet Service**: This service exposes a rich API for filtering and querying against the data sources that store our Twitter datasets. Currently, this service is wired to connect to our Solr cluster, but we have plans to expand its API to allow access to tweets stored in different data stores, including our underlying Cassandra cluster. The idea is to allow clients to identify the data source best suited to handle the queries they want to submit.

While each service is independently accessible, the QueryDispatcher service is the primary access point for a majority of the current set of workflows used by our analysts. The key extension point for this service is the query components that encapsulate the knowledge of how to work with a particular data source. If an analyst or developer identifies a new target data store, they can write a new query component, submit a pull request on the repository for the QueryDispatcher service, wait for their change to be committed, and then, finally, their new component manifests as a new URI (i.e. endpoint) that can be handled by the QueryDispatcher service. New endpoints, in turn, mean new ways of querying the data stored in EPIC Analyze and EPIC Collect that are then available to all users of the system. To date, the next version of EPIC Analyze has the following query components (a.k.a. strategies): a TweetQueryStrategy to retrieve tweets, a DatasetQueryStrategy to retrieve metadata associated with our datasets, and an AnalystQueryStrategy to retrieve metadata about an analyst's profile. These new services can be used to implement the query life cycle shown in Section 4. The new query life cycle is shown in Fig. 7.

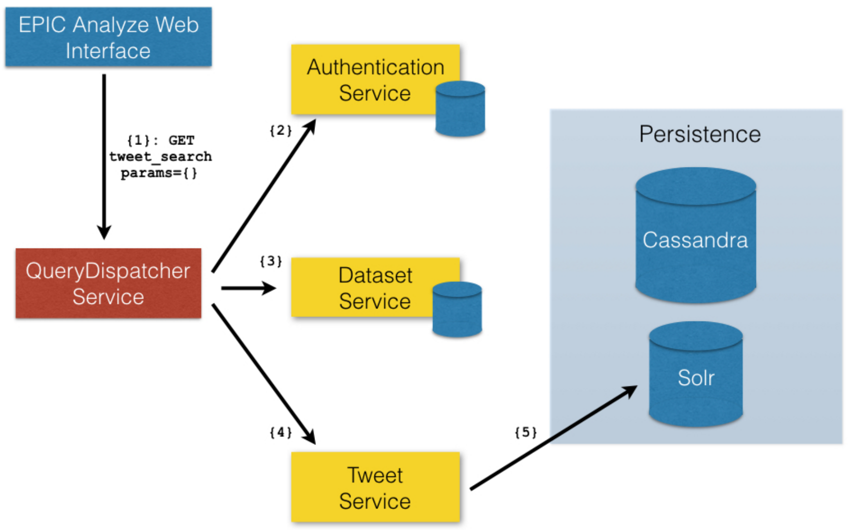


Fig. 7. EPIC Analyze Query Request-Response Life Cycle using Microservices.

(i) A request to query an Epic Analyze dataset is initiated; the payload contains the query and the credentials of the analyst making the request. This request is sent to the QueryDispatcher service via HTTP.
(ii) The QueryDispatcher service invokes the TweetQueryStrategy, which requires authentication. Therefore, it first authenticates the credentials by sending a request to the Authentication Service, which checks its database for a record that matches those credentials and returns an access token or denies the request.

(iii) If the request is authenticated, the QueryDispatcher then checks the request to see if there is a dataset specified. If one exists, then the QueryDispatcher will make a request to the Dataset service asking for that dataset's metadata.

(iv) Equipped with the metadata of the Dataset, the QueryDispatcher finally makes a call to the Tweet service, where the parameters of the current request are converted to a proper query string for the target data store.

(v) The Tweet Service will sanitize the incoming query and then (in this instance) send the request to Solr, which will return the results associated with that query. The QueryDispatcher service receives paginated Tweet data which it will return in the format requested by the client.

As mentioned above, a client could have alternatively made service calls to the Authentication, Dataset, and Tweet services all on its own, but that increases the amount of code that needs to go into the client including error-handling for each call and knowledge of where those services are located. The use of the QueryDispatcher service isolates that knowledge into a single service that helps to loosen the coupling between clients and EPIC Analyze.

We now describe how EPIC Analyze had limitations with respect to the three characteristics identified above and how our migration to a microservices-based software architecture allowed us to address them.

6.3 Extensibility

The monolithic version of EPIC Analyze would struggle to handle the addition of new features not related to querying and/or manipulating Project EPIC's collected datasets. If analysts wanted to submit queries on tweets currently streaming in via Twitter's Streaming API, it would require new server-side components to access that API, store the streaming tweets in EPIC Collect, issue queries on the most recent "window" of tweets and a new user interface to display tweets that match the current query. Such an enhancement *could* be achieved but it would require a significant amount of work and re-design and would, no doubt, lead to an increase in complexity for the entire application.

With the microservices-based version of EPIC Analyze, we have found that feature enhancements can be added in a straightforward manner by spinning up a new service. We have begun work on designing a new service to support the use case above. The service can accept a query that then gets submitted to the Twitter Streaming API. The service stores the last thirty seconds of tweets in memory and can apply a submitted query to that group of tweets and return all tweets that match the query. We have yet to develop a user interface that accesses that new service but the important point is that none of the existing services and user interfaces of EPIC Analyze had to change as a result of spinning up this new service. Instead, our users can continue to use EPIC Analyze as they normally do while our development team works on adding a new user interface that can handle the display of streaming tweets and (eventually) integrating that display into the existing user interface. The microservices approach enables an environment that allows research analysts and developers to work together to jointly develop new enhancements to the analysis environment much more easily than in the environment provided by the monolithic software architecture.

6.4 Independence

What has become increasingly prevalent in our research lab is the level of autonomy that each researcher has to pursue his or her own research questions. With this autonomy comes heterogeneity in technology, technique, and choice of research methods across the fields of natural language processing, social and information science, and software engineering. Therefore, our chief goal is to design and maintain an infrastructure that supports analyst workflows but also "gets out of the way" when analysts are exploring and analyzing data in ways that fall outside the standard features discussed in Section 5.

We want to enable analysts from all disciplines to make use of EPIC Analyze in whatever way is most efficient. At the same time, we do not want to prevent analysts from building their own tools, applications, or services. Using a microservices approach, we can improve upon the state of affairs by effectively exposing functionalities that were once confined "within" the analysis environment. This exposure achieves two goals: first, it encourages anyone in our research environment to build the tools that they need—using any programming language, library, and technology they select—to answer their research questions, while also leveraging tools that already exist; and second, it encourages knowledge sharing and tool reuse by allowing common services to be developed and shared. We envision that this level of independence will create infrastructural transparency, allowing the infrastructure to fade into the background and allow research goals and workflows to take center stage.

6.5 Accessibility

A third limitation of the monolithic approach is the lack of accessibility, which we define to mean the ease with which users of the infrastructure can access and use our analysis tools. Generally, this is quantified by the number of user interfaces available to access the system. Our work on the original paper [12] examined issues of query expressivity and visibility within EPIC Analyze's user interface, issues important to address in the context of visually analyzing datasets using a graphical interface.

However, microservices easily expand the notion of "interface" to one that is accessible via a REST-based API. In this way, querying interfaces for web-based data-intensive software systems have the ability to service a wider range of clients than what is possible with an HTML-based interface within a web browser. Since many of our analysts are skilled developers, this ability provides them with new ways to interact with EPIC Analyze. Microservices were a natural technique for providing this type of interface. The QueryDispatcher service becomes a new interface for accessing our Twitter datasets outside of the browser-based user interface. Supporting popular formats such as JSON and CSV are easily accomplished by new services or extensions to existing services and this, in turn, allows our analysts to work efficiently on their own projects and take advantage of what the infrastructure has to offer.

6.6 Summary

In this section, we discussed the next version of EPIC Analyze which has been migrated to make use of microservices rather than rely on a monolithic approach to implementing a web application. These services have allowed us to increase EPIC Analyze's support for extensibility, accessibility, and independence. The new service based approach significantly simplifies the implementation of the query life cycle that was discussed in Section 4. This new version of EPIC Analyze positions us well to add new feature enhancements that directly support the analysis workflows of Project EPIC's analysts. That, in turn, increases our ability and capacity to perform new types of crisis informatics research.

7 Related Work

The explosion of data in diverse application domains has led to a variety of research efforts developing and evaluating software infrastructures that provide processing and analysis at scale. Additionally, emergency management has seen a surge of analytical tools and dashboards that operate on such infrastructures in order to provide fast and timely relief during disaster events. We discuss related work in big data research while highlighting efforts in crisis data management systems. To properly ground our work, we describe recent research efforts in microservices and the design of big data software architectures as well as work that addresses the design of user interfaces for collecting, wrangling, and querying big data.

7.1 Big Data Architectural Design

Software architectural design plays an important role in producing scalable, reliable, and usable data-intensive systems. It is important to identify the right architecture and the combination of middleware and persistence solutions to efficiently support data collection, storage, and analysis. To handle challenges of high volume, variety, and velocity, new approaches have presented a departure from building monolithic applications to developing distributed software services. This migration is in part due to the emergence of service-oriented architecture and microservices and their promise for improved interoperability. As we discussed in this paper, EPIC Analyze has followed a similar migration from a monolithic design of its web application to one based on microservices.

Due to the heterogeneity of data in today's modern information environment, web applications need to integrate different database layouts and engines, including a range of NoSQL technologies such as graph databases, document data stores, and columnar datastores. In [33], Viennot et al. set their sights on addressing this problem with their work on Synapse. Synapse is a framework for developing complex web applications with more than one type of data store; Synapse is based on a microservices architecture. Synapse leverages the standard MVC framework by allowing developers to declaratively extend their applications controllers, models, and ORMs to share data across services in a simple publish/subscribe model through Synapse's API. Synapse, developed on top of Ruby on Rails, supports data replication among a wide range of SQL and NoSQL databases, including MySQL, Oracle, PostgreSQL, MongoDB, Cassandra, Neo4j, and Elastic Search. Synapse has been used as a scalable cross-database replication system in a production environment in support of many data-driven web services, solving integrations problems while supporting agile development. Our motivation for adopting microservices in our own work is complementary to what is seen in Synapse. Rather than focusing exclusively on access to data sources, we see microservices as a way to encapsulate user-level functionality that can then be implemented, tested, and deployed independently from other core features of the system.

With respect to managing metadata, Dey et al. have proposed using a metadata-as-a-service component within an enterprise environment [32]. Metadata-as-a-Service can uniquely identify datasets living in heterogeneous stores, describing their schema, size, and accessibility.

The authors believe that such services will help workers in an enterprise environment find the data they need to re-use rather than re-build. As discussed in Section 6, we adopted this approach to metadata within EPIC Analyze for access to datasets and user profiles.

In [34], a microservices architecture pattern is tested and analyzed via a case study. The authors created an enterprise application using both the monolithic and microservices-based styles. Each application was implemented using the Play framework; both versions were deployed using Amazon's cloud services. The authors conducted evaluations of both applications looking at performance, development methodology, deployment, operation, and adoption by end users. The performance of both applications was similar with the monolithic architecture slightly faster with respect to average response time. However, the microservices architecture led to a 17% reduction of infrastructure costs. The authors point out that microservices do introduce some complexity into a system with respect to distributed systems concerns (such as timeouts) and the need for separately configuring each microservice, but it also introduces flexibility with respect to the ability to use different technologies for each microservice and the ability to deploy them onto hardware best suited for their needs. The authors of this study conclude that migration to microservices architecture should not be seen as a single project but rather a long term strategy that should be applied in an incremental fashion.

In [35], a cloud-based, collaborative, big data analytics platform is proposed. The platform provides multi-tenancy and allows developers to focus on the development of services while sharing existing resources such as other services, data, and algorithms. Developers, data owners, and data scientists can collaboratively work by jointly managing large datasets and designing/improving analysis algorithms without impacting each other's work all the while sharing resources such as compute cycles, memory, and disk. Moreover, a web service portal is provided to facilitate a user's interaction with managing data and accessing development tools. The analytics portal of the platform provides the following services: 1) a Query Editor to manage hive and relational database queries; 2) a Data Browser for viewing information in HBase and Hive's metastore; 3) a Workflow Designer that provides a web-based service modeling tool that allows users to develop algorithms; and 4) a Job Browser and Monitoring Tool to monitor the platform's resources. In [35], the authors demonstrate that a service to analyze data from a closed circuit video monitoring system could be created and deployed. In much the same vein, our work revolves around a collaborative data analysis system that has evolved in the past few years to become a shared platform upon which research analysts and developers can build and share tools for data analysis on a large scale.

Finally, in [36], Le et al. present a collaborative effort to redesign a large software system to make use of microservices. The large system was a web-based application that provided services in support of the Nevada Research Data Center. This center hosts collaborative research projects consisting of scientists, engineers, educators, and technicians with the goal of increasing the amount of renewable solar energy in Nevada. Over time, their system was being asked to handle a wide range of new requirements that involved processing new types of data. They discovered that it was nearly impossible to respond to these requests in an efficient manner with the original design of their system, a monolithic web app build in Django.

Looking for an architecture that provides scalability, reliability, and maintainability, the authors settled on microservices and they present the benefits they gained by migrating from their monolithic system to a service-based environment. The new architectures consists of five modules—Person, Project, System, Component, and Service Entry—each is an independent service, built using the Flask microframework, all hosted separately in five Flask servers and connected to five PostgreSQL databases. The system has a front-end website for easy access to these data-driven services, but also these services can be reached via an API that can be accessed by command-line tools. Le et al. assert that microservice-based architecture is the best solution for a distributed service architecture of autonomous modules that are easy to manage and evolve. It helped them to develop independent services while managing the relationships among them. One key requirement that microservices satisfied was that the NRDC platform had to remain active as the migration took place. Since microservices can be spun up individually, it was possible to turn off an endpoint in the old monolithic application and then in the web front-end simply redirect calls to that endpoint to the new microservice instance. In this way, they could maintain service while incrementally migrating to the new approach. We had less services to deal with when performing our migration to microservices with EPIC Analyze but we realized many of the same benefits as reported by [36].

7.2 Crisis Informatics and Social Media Analysis

Social media platforms generate large amounts of data all day, every day. However, work in disaster studies has shown that natural disaster events generate unprecedented volumes of data that challenge our notion of scalability and reliability in crisis software systems. The information deluge generated via social media channels create challenges with respect to data handling all throughout the collection and analytics pipeline. In the crisis informatics space, there is much work to be done with respect to software design for members of the public to coordinate and collaborate online, such as reporting on and matching missing domestic animals [37] or identifying and verifying image authenticity [38]; creating data dashboards for emergency management personnel [39, 40]; and mobilizing online communities to perform crowdwork tasks [15, 17].

Fortunately, much work has already been performed in the area of big data analytics targeting crisis data. Cameron et al. [39] presented an emergency situational awareness platform to help crisis coordinators in the Australian Government Crisis Coordination Centre detect incidents and identify event-related reports published by the public in Twitter. The platform, built using simple databases, a Java-based messaging system, and web services, provides a new source of data for an existing emergency management and crisis coordination system. The authors found social media data needs to be corroborated with traditional sources of information to support emergency managers with enhanced situational awareness. Tin et al. [41] also present an integrated framework for big data analytics to analyze disaster events. The framework developed a useful information prediction scheme based on co-occurrence theory and Markov chain modeling to help improve the automatic detection of disaster events via social media. The framework simulated results from data collected from social media platforms in the 2011 Japan Earthquake and Tsunami event.

In our current work, EPIC Analyze provides filtering, searching, sorting, sampling, and annotation services on the top of Project EPIC's crisis datasets for a similar purpose to the systems developed by Cameron et al. and Tin. Our work focused more on the software architectural concerns of scalability, reliability, and extensibility and has been used more for the internal study of the events that Project EPIC has focused on and less on the use in emergency management contexts or in the prediction of disaster events. Project EPIC makes use of traditional media and social media to identify events and then our 24/7 data collection service can be configured to collect data on them. In situations where we miss the start of the event, we make use of the Twitter Search API and Gnip's PowerTrack API to go back in time to gain access to the tweets that we missed.

7.3 Interface Design for Big Data Systems

Providing users with interfaces to explore and analyze large amounts of data represents another class of challenges in the design and development of data-intensive software systems. Software engineering researchers and practitioners face challenges in capturing, processing, integrating, analyzing, and archiving big data. To add to that burden, the goal for human-centered computing research in this domain is to put the power of big data systems into the hands of non-technical users [42]. Creating intuitive, flexible, and extensible user interfaces that allow users to pull from structured and unstructured data sources, query and analyze the data, and make more informed claims about the data, are the objectives of big data interfaces. Users of such interfaces do not need to become familiar with big data frameworks—e.g. MongoDB, Redis, Cassandra, Spark, etc.—but need to have confidence that the systems built on top of them are reliable and efficient; otherwise they may choose to stay away from working with big data or seek to use other technologies that do not have the same capacity for scale and thus be prevented from taking advantage of the benefits of big data analysis.

Systems like Wrangler [43] and Google Refine (`http://openrefine.org/`) decrease the amount of work required to transform data; this allows those not proficient in programming to work in this space, converting large data sets into the format they need. For large time series data sets, there are several advances in diverse domains. The LifeFlow system [44] aggregates event data from hospital visits and room transitions and visualizes them to identify problems in triaging or resource allocation. Splunk (`http://www.splunk.com`) is a commercial data analysis platform for working with time series data, providing a pipe- based textual language to manipulate data. There are shortcomings to these tools, however. Wrangler, Refine, and LifeFlow do not provide support for user-generated annotations and do not support collaboration. Splunk's programming language is flexible but has a significant learning curve, especially for analysts not familiar with the pipe-and-filter software architectural style.

One must also appreciate the infrastructure that is built behind such interfaces; often the design of the software infrastructure itself shapes the look and feel of the interface (as we discussed in Section 4). With data analysis platforms, it is not good enough to display a web page with filters for querying a database. Oussalah et. al [45] presents a web-based analysis environment that ties in semantic and spatial analyses of tweets in addition to straightforward search capabilities. This work inspired aspects of the design of EPIC Analyze, especially with respect to providing a suite of integrated services for the end user.

Another effort in big data user interfaces is presented in [46]. This system supports user exploration of large data sets by constructing ad hoc queries in a lightweight web-based framework called BigExcel. The framework consists of three tiers for handling user interactions with large data sets, constructing queries to explore data sets, and managing the underlying infrastructure. The goals pursued from developing the framework is to provide for quick and easy analytics environment for end users. Their work is relevant to EPIC Analyze as it provides big data accessibility to researchers, allowing them to explore large sets of data and create ad hoc queries on different datasets.

8 Future Work and Discussion

The microservices-based version of EPIC Analyze is operational and will soon replace the current version as the one made available to our analysts for their daily work. The EPIC Analyze web application has been modified to direct its calls to the QueryDispatcher service and work is now focused on extending the number of query components integrated into it. Once available, we will continue to monitor the activity of our analysts to see how well the new version of EPIC Analyze is able to meet their needs and how it shifts our own ability to develop new features for the environment.

In the future, our work will shift to address a number of topics including 1) real-time analytics, 2) support for collaboration, 3) integration of machine learning and network analysis techniques, and 4) visualization.

Real-time analytics is a topic of increasing popularity; we believe that support for real-time analysis of Twitter data streams would be an excellent addition to EPIC Analyze's capabilities. It will allow analysts to ask research questions in real-time while an event is underway and look for live social interactions that provide insight into those questions.

Support for collaboration within a big data analytics environment represents a wealth of open research questions with respect to designing tools, interfaces, and techniques to increase productivity in a variety of distance- and time-based settings. In crisis informatics, collaboration is important in working with remote third- party collaborators or side-by-side with other researcher in the local research group. One feature that we will be exploring within this space is providing user-defined coding schemes to support qualitative tagging activities over our Twitter datasets.

A frequent feature request for EPIC Analyze from our analysts is to support more statistical and computational algorithms on top of our datasets. Before the microservices-based architecture, these requests were difficult to handle as support for them would require a careful redesign of the existing components. Machine learning classification is one technique that would greatly support analytical workflows by automatically tagging tweets that may represent interesting behaviors, such as requests for help in a disaster situation [47]. Furthermore, one of our analysts desires the ability to apply network algorithms (graph-based analysis techniques) on our Twitter datasets to identify, for instance, people who are local to a disaster event and may need different types of information from those watching the event from afar.

Finally, visualization is an exciting frontier for research in data-intensive software systems. To date, we are only able to offer a visualization timeline to view volume of tweets over time for a specific dataset. What would be more interesting, however, is to provide the right visualization with respect to the queries being made. For example, if an analyst using EPIC Analyze wants to filter tweets based on geo-location, then EPIC Analyze can determine that a map-based visualization is more appropriate to render than a timeline. Map visualization operations should also be supported with gestures, such as clicking and dragging, hovering, and double-clicking. Support for these operations will provide a richer user experience for analysts when analyzing geotagged data.

9 Conclusions

In this paper, we have presented our work on designing user interfaces and APIs for large-scale data analysis environments. We reported on challenges we faced designing a user interface for the browser of EPIC Analyze, our analysis platform for crisis informatics research. Our work—influenced by our strong human-centered computing perspective—took advantage of feedback provided by seven Project EPIC analysts who regularly work with crisis data sets. Their feedback led to a number of improvements that allowed EPIC Analyze to become a tool they use on a daily basis. We focused on a challenging user interface design puzzle—drilling down into a large data set—and showed how the solution required a new approach to implementing queries on the server to support new elements in the user interface.

We then discussed how the solution to our user interface challenge while effective was too tightly coupled and would be difficult to extend to other use cases. We identified microservices as a promising avenue for exploring how to solve this new problem and then described how we redesigned and re-factored EPIC Analyze to make use of this new paradigm. Our goal now is to extend EPIC Analyze to support a broader range of analysis with the integration of new tools via microservices.

Developers have relied upon decades of work to provide them with sophisticated development environments and interfaces that support the development of desktop and mobile applications. However, for big data applications, we have very little tool support in contrast. Indeed, the most common way a developer interacts with the systems and frameworks that are used to build data intensive systems is the humble command-line terminal [1]. In the next decade, developers need to design and implement an entire ecosystem of support tools that make generating data intensive systems a more straightforward task. This in turn will have benefits downstream allowing the developers of these systems to offer better interfaces for analysts to query large data sets, identify patterns and relationships hiding in the data, and suggest how to further wrangle the data as necessary [7].

Acknowledgements

This material is based upon work sponsored by the NSF under Grant IIS-0910586. We would like to thank the seven analysts who participated in our think-aloud sessions.

References

1. K. M. Anderson, “Embrace the challenges: Software engineering in a big data world,” in *First International Workshop on Big Data Software Engineering, Part of the 2015 International Conference on Software Engineering*, pp. 19–25, May 2015.
2. L. Palen, K. M. Anderson, G. Mark, J. Martin, D. Sicker, M. Palmer, and D. Grunwald, “A vision for technology-mediated support for public participation & assistance in mass emergencies & disasters,” in *ACM-BCS Visions of Computer Science*, April 2010. Article 8. 12 pages.
3. K. M. Anderson and A. Schram, “Design and implementation of a data analytics infrastructure in support of crisis informatics research (nier track),” in *International Conference on Software Engineering*, pp. 844–847, May 2011.
4. K. M. Anderson, A. Schram, A. Alzabarah, and L. Palen, “Architectural implications of social media analytics in support of crisis informatics research,” *IEEE Bulletin of the Technical Committee on Data Engineering*, vol. 36, pp. 13–20, September 2013.

5. A. Schram and K. M. Anderson, "MySQL to NoSQL: Data modeling challenges in supporting scalability," in *Systems, Programming Languages, and Applications: Software for Humanity*, pp. 191–202, October 2012.
6. K. M. Anderson, A. A. Aydin, M. Barrenechea, A. Cardenas, M. Hakeem, and S. Jambi, "Design challenges/solutions for environments supporting the analysis of social media data in crisis informatics research," in *Hawaii International Conference on System Sciences*, pp. 163–172, IEEE, January 2015.
7. D. Fisher, R. DeLine, M. Czerwinski, and S. Drucker, "Interactions with big data analytics," *Interactions*, vol. 19, pp. 50–59, May + June 2012.
8. L. Palen, S. Vieweg, J. Sutton, S. Liu, and A. Hughes, "Crisis informatics: Studying crisis in a networked world," in *Third International Conference on E-Social Science*, October 2007. 10 pages.
9. A. Sarcevic, L. Palen, J. White, K. Starbird, M. Bagdouri, and K. M. Anderson, "'Beacons of hope' in decentralized coordination: Learning from on-the-ground medical twitterers during the 2010 haiti earthquake," in *Computer Supported Cooperative Work*, pp. 47–56, February 2012.
10. A. L. Hughes, L. St. Denis, L. Palen, and K. M. Anderson, "Online public communications by police & fire services during the 2012 hurricane sandy," in *Human Factors in Computing Systems*, pp. 1505–1514, April 2014.
11. L. Huang, K. Starbird, M. Orand, S. Stanek, and H. Pedersen, "Connected through crisis: Emotional proximity and the spread of misinformation online," in *Computer Supported Cooperative Work*, pp. 969–980, March 2015.
12. M. Barrenechea, K. M. Anderson, A. A. Aydin, M. Hakeem, and S. Jambi, "Getting the query right: User interface design of analysis platforms for crisis research," in *International Conference on Web Engineering*, pp. 547–564, June 2015.
13. C. Hagar, "Crisis informatics: Perspectives of trust—is social media a mixed blessing?," *iSchool Student Research Journal*, vol. 2, no. 2, 2013.
14. K. Starbird and L. Palen, "'Voluntweeters': Self-organizing by digital volunteers in times of crisis," in *Human Factors in Computing Systems*, pp. 1071–1080, May 2011.
15. J. White, L. Palen, and K. M. Anderson, "Digital mobilization in disaster response: The work & self-organization of on-line pet advocates in response to hurricane sandy," in *Computer Supported Cooperative Work and Social Computing*, pp. 866–876, February 2014.
16. K. Starbird, L. Palen, A. Hughes, and S. Vieweg, "Chatter on the red: What hazards threat reveals about the social life of microblogged information," in *Computer Supported Cooperative Work*, pp. 241–250, February 2010.
17. L. Palen, R. Soden, T. J. Anderson, and M. Barrenechea, "Success and scale in a data-producing organization: The socio-technical evolution of openstreetmap in response to humanitarian events," in *Human Factors in Computing Systems*, pp. 4113–4122, April 2015.
18. J. Bargas-Avila and K. Hornbæk, "Foci and blind spots in user experience research," *Interactions*, vol. 19, pp. 24–25, November + December 2012.
19. C. McTaggart, "Analysis and implementation of software tools to support research in crisis informatics," Master's thesis, University of Colorado, 2012. 65 pages.
20. J. Laconich, J. José, F. Casati, and M. Marchese, "Social spreadsheet," in *Web Engineering* (F. Daniel, P. Dolog, and Q. Li, eds.), vol. 7977 of *Lecture Notes in Computer Science*, pp. 156–170, Springer Berlin Heidelberg, 2013.
21. C. Lewis and J. Rieman, *Task-centered User Interface Design: A Practical Introduction.* Department of Computer Science, University of Colorado, Boulder, 1993. 170 pages.
22. B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *IEEE Symposium on Visual Languages*, pp. 336–343, September 1996.
23. A. A. Aydin and K. M. Anderson, "Incremental sorting for large dynamic data sets," in *First IEEE International Conference on Big Data Computing Service and Applications*, pp. 170–175, March + April 2015.
24. C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: A not-so-foreign language for data processing," in *International Conference on the Management of Data*, SIGMOD

'08, (New York, NY, USA), pp. 1099–1110, ACM, June 2008.
25. M. Fowler and J. Lewis, "Microservices," March 2014. `http://martinfowler.com/articles/microservices.html`.
26. I. Graham, *Object-Oriented Methods: Principles & Practice*. Addison-Wesley, 2000. 832 pages.
27. C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. Addison Wesley, 2002. 624 pages.
28. R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. 162 pages.
29. P. Dix, *Service-Oriented Design with Ruby and Rails*. Addison-Wesley Professional, 2010. 320 pages.
30. N. Marz and J. Warren, *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications, 2015. 328 pages.
31. T. Erl, C. Gee, P. Chelliah, J. Kress, H. Normann, B. Maier, L. Shuster, B. Trops, T. Winterberg, C. Utschig, and P. Wik, *Next Generation SOA: A Concise Introduction to Service Technology & Service-Orientation*. Pearson Education, 2014. 185 pages.
32. A. Dey, G. Chinchwadkar, A. Fekete, and K. Ramachandran, "Metadata-as-a-service," in *International Conference on Data Engineering Workshops*, pp. 6–9, IEEE, April 2015.
33. N. Viennot, M. Lécuyer, J. Bell, R. Geambasu, and J. Nieh, "Synapse: A microservices architecture for heterogeneous-database web applications," in *European Conference on Computer Systems*, pp. 1–16, April 2015.
34. M. Villamizar, O. Garces, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil, "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud," in *Computing Colombian Conference*, pp. 583–590, September 2015.
35. K. Park, M. C. Nguyen, and H. Won, "Web-based collaborative big data analytics on big data as a service platform," in *International Conference on Advanced Communication Technology*, pp. 564–567, IEEE, July 2015.
36. V. D. Le, M. M. Neff, R. V. Stewart, R. Kelley, E. Fritzinger, S. M. Dascalu, and F. C. Harris, "Microservice-based architecture for the NRDC," in *International Conference on Industrial Informatics*, pp. 1659–1664, July 2015.
37. M. Barrenechea, K. M. Anderson, L. Palen, and J. White, "Engineering crowdwork for disaster events: The human-centered development of a lost-and-found tasking environment," in *Hawaii International Conference on System Sciences*, pp. 182–191, IEEE, 2015.
38. A. Popoola, D. Krasnoshtan, A.-P. Toth, V. Naroditskiy, C. Castillo, P. Meier, and I. Rahwan, "Information verification during natural disasters," in *International Conference on the World Wide Web*, (New York, NY, USA), pp. 1029–1032, ACM, April 2013.
39. M. A. Cameron, R. Power, B. Robinson, and J. Yin, "Emergency situation awareness from twitter for crisis management," in *21st International Conference Companion on the World Wide Web*, pp. 695–698, ACM, April 2012.
40. L. Han, S. Potter, G. Beckett, G. Pringle, S. Welch, S.-H. Koo, G. Wickler, A. Usmani, J. L. Torero, and A. Tate, "Firegrid: An e-infrastructure for next-generation emergency response support," *Journal of Parallel and Distributed Computing*, vol. 70, no. 11, pp. 1128–1141, 2010.
41. P. Tin, T. T. Zin, T. Toriu, and H. Hama, "An integrated framework for disaster event analysis in big data environments," in *International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pp. 255–258, IEEE, October 2013.
42. J. Heer and S. Kandel, "Interactive analysis of big data," *Crossroads: The ACM Magazine for Students*, vol. 19, pp. 50–54, September 2012.
43. S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer, "Wrangler: Interactive visual specification of data transformation scripts," in *Human Factors in Computing Systems*, pp. 3363–3372, May 2011.
44. K. Wongsuphasawat, J. A. G. Gómez, C. Plaisant, T. D. Wang, M. Taieb-Maimon, and B. Shneiderman, "Lifeflow: Visualizing an overview of event sequences," in *Human Factors in Computing Systems*, pp. 1747–1756, May 2011.
45. M. Oussalah, F. Bhat, K. Challis, and T. Schnier, "A software architecture for twitter collection,

search, and geolocation services," *Knowledge-Based Systems*, vol. 37, pp. 105–120, 2013.
46. M. A. Saleem, B. Varghese, and A. Barker, "Bigexcel: A web-based framework for exploring big data in social sciences," in *International Conference on Big Data*, pp. 84–91, IEEE, October 2014.
47. S. Verma, W. Corvey, S. Vieweg, J. Martin, L. Palen, M. Palmer, A. Schram, and K. M. Anderson, "NLP to the rescue?: Extracting 'situational awareness' tweets during mass emergency," in *International AAAI Conference on Weblogs and Social Media*, pp. 385–392, July 2011.