Design and Implementation of a Data Analytics Infrastructure in Support of Crisis Informatics Research (NIER Track)

Kenneth M. Anderson
University of Colorado
Department of Computer Science
Boulder, CO 80309-0430
kena@cs.colorado.edu

Aaron Schram
University of Colorado
Department of Computer Science
Boulder, CO 80309-0430
aaron.schram@colorado.edu

ABSTRACT

Crisis informatics is an emerging research area that studies how information and communication technology (ICT) is used in emergency response. An important branch of this area includes investigations of how members of the public make use of ICT to aid them during mass emergencies. Data collection and analytics during crisis events is a critical prerequisite for performing such research, as the data generated during these events on social media networks are ephemeral and easily lost. We report on the current state of a crisis informatics data analytics infrastructure that we are developing in support of a broader, interdisciplinary research program. We also comment on the role that software engineering research plays in these increasingly common, highly-interdisciplinary research efforts.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures; D.2.13 [**Software Engineering**]: Reusable Software

General Terms

Design

Keywords

Crisis Informatics, Software Frameworks, Scalability

1. INTRODUCTION

Crisis informatics is an emerging research area that studies how information and communication technology (ICT) is used in emergency response. An important and newly studied branch of this research is how members of the public make use of communication technology to aid them during mass emergencies [1]. This is our focus. Data collection and analytics during crisis events is a critical prerequisite for performing research in this field of crisis informatics as the data that are generated during these events on social media networks are ephemeral and easily lost. Yet it is just this type of information that is needed to understand the online information-seeking and sensemaking behaviors that members of the public engage in during times of crisis [3].

We report here on the current state and future directions of a data analytics infrastructure that we are developing in support of a broader, interdisciplinary research program in crisis informatics [2]. In this effort, software engineering research plays a critical role that helps bridge the work of researchers from highly technical domains, such as natural language processing, with researchers in human-centered computing (HCC) and the sociology of disaster. For instance, we must strike a balance in producing an infrastructure that allows new services from our more technical colleagues to be added in a straightforward manner while also meeting the stringent methodological requirements for empirical data collection by our HCC and behavioral studies colleagues. (If certain constraints related to the sampling of data streams are not met, it is difficult for our colleagues to extract statistically-significant results from the collected data.) Furthermore, we must strike that balance while endeavoring to meet our own research goals related to finding the right combination of techniques and frameworks to create a production-class data analytics infrastructure that is flexible, scalable, reliable and highly concurrent. How do we achieve these characteristics while providing the set of services needed to perform useful crisis informatics research?

All of these characteristics need to be met to have a chance of producing the comprehensive data sets our colleagues require. To demonstrate this, consider the challenges we faced while collecting data from Twitter during the three weeks after the January 12, 2010 earthquake that occurred 16 miles west of Port-au-Prince, Haiti. Our data collection service needed to be reliable since we had to collect data from Twitter for three continuous weeks to ensure that we did not miss relevant data points as Twitter does not guarantee that tweets will be accessible after they are generated.

The data collection infrastructure had to be scalable—ultimately we collected 4.25M tweets from 1.25M unique users matching the search terms our colleagues specified; they then identified a subset of 80K users that they wanted additional information on—specifically their complete tweet streams, whether or not each tweet contained the search terms—requiring collection of an additional 75.8M tweets. The combined data set consumed 100 GB of disk space for

ICSE '11, May 21–28, 2011, Waikiki, Honolulu, HI, USA

a single crisis event; our team has since collected data on tens of events that occurred in 2010 for a combined total of 1B tweets consuming 100s of GBs of disk space.

The infrastructure must be highly concurrent to enable high-volume data collection (our project obtained five whitelisted I.P. addresses and one whitelisted account from Twitter to allow us to collect a theoretical maximum of 500M tweets per day) and to analyze the collected data efficiently. All of our code is multithreaded and designed to adapt to multicore machines, spawning more threads based on the number of cores available.

Finally, our infrastructure must be flexible enough for us to quickly adapt to new requirements that arise during data collection which can rapidly change during emergency response. For instance, during the September 2010 wildfires in Boulder, Colorado, our team decided to collect data on the changes to the "friends and followers" of Twitter users who generated tweets using the keywords selected for that event. This type of data is useful, for example, to develop algorithms that can identify influential users of social media during a crisis event. We had previously built a service to collect the friends and followers of a single Twitter user, but in this event, we needed it integrated into the infrastructure for frequent and regular data collection of the social graph data as those social graphs quickly changed. The flexibility afforded by our software architecture allowed us to integrate that service and its associated application logic into the infrastructure in under a day without disrupting the preexisting data collection services.

We now provide details on the architectural choices made to achieve these characteristics. It is important to note that though we designed and built this infrastructure around existing production-class frameworks, there are many non-obvious customizations we made to these frameworks to achieve our goals. When these customizations are combined with the way we designed the conceptual layers of the infrastructure and the way we create applications on top of our services, our approach represents a new direction to developing production software systems in research contexts. One lesson from this work is that non-scalable, non-reliable prototypes in multidisciplinary research settings are a thing of the past. The constraints of the problem domains associated with the cross-cutting research programs demanded by new frontiers of science no longer allow for simplistic approaches to the development of research prototypes.

2. DATA ANALYTICS INFRASTRUCTURE

The logical architecture of our data analytics framework for crisis informatics research is shown in Fig. 1. Though our current set of services focuses on collecting and analyzing data from Twitter, our infrastructure's capabilities can be expanded by simply adding additional services that interact with other external data sources. The functionality currently provided by our infrastructure is to 1) collect tweets that match a given set of keywords, 2) collect all the tweets generated by a given set of users and 3) capture a snapshot of the social graph surrounding a given Twitter user for a given number of hops. We make full use of all of the available Twitter APIs including the Streaming API to capture tweets as they occur, the Search API to search back in time for tweets that match a keyword, and the REST API to access user profiles and their associated tweets.

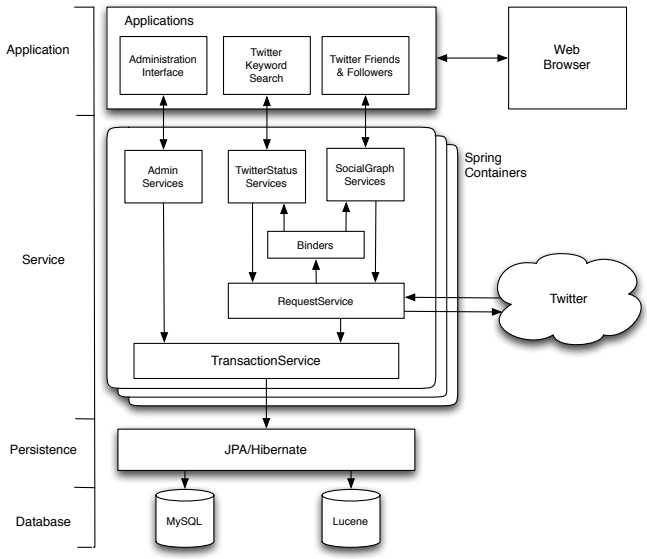

Figure 1: The logical architecture of our data analytics infrastructure for crisis informatics research. Note: some information flows have been elided to avoid visual clutter. For instance, services that access the transaction service to interact with the database can both send and receive information.

2.1 Infrastructure Components

The infrastructure is built using production-class software frameworks (Spring & Spring MVC, Hibernate and JPA) and infrastructure components (Tomcat, MySQL and Lucene).[1] Spring provides the ability to create a testable and reusable code base via the use of their dependency injection framework. Most of the Java classes that make up our services conform to the Spring bean definition and thus get instantiated and managed by the Spring container.

We use Spring to configure and manage our database transactions throughout all of our applications. We use a mix of declarative Spring transactions using the *@Transactional* annotation but in a custom transaction callback that allows us to fine-tune transaction boundaries depending on the needs of each service. We use this ability to customize transaction boundaries to allow, for example, our social graph service to knit together a set of Twitter users into a community while the data collection service is concurrently updating the set of tweets associated with the community's users. The transaction callback allows us to specify a reusable atomic grouping of functionality while ensuring the social graph service is not holding onto stale references of a large set of users when only a few are being accessed.

Spring MVC is used as our web framework. We made this choice to ensure that all of our web applications are "container agnostic" using Spring libraries to handle such things as transactions, content negotiation and security. This type of flexibility allows our applications to be deployed in a wide variety of server environments. We also provide a REST-based web service interface to our services and these are written using Spring MVC as well. This choice maximizes

[1]References to the software frameworks cited in the paper are easily found via an Internet search.

software reuse since the same Spring services can respond to requests from web service clients as well as browser-based clients—with Spring automatically handling the on-the-wire formats that clients can generate or consume.

All of the Java objects that represent the shared "model" of our services conform to JPA 2.0. The model is shared in the sense that a Twitter user discovered by our social graph service and stored in the database creates an object that is updated when our data collection service discovers a tweet that was generated by that Twitter user. That is, both services access and update the same set of model objects stored in the database. The fact that all of our model objects conform to JPA 2.0 allows our infrastructure to be deployed on a wide variety of RDBMSs without any refactoring of our code base. To use a different persistent storage provider requires a straightforward update to Spring's configuration files. We chose Hibernate as our JPA provider. Our existing services are heavy on inserts; that is, they spend more time adding new objects to the database than they do reading the information contained in the database. Initially, we were concerned that Hibernate's object-relational mapping capabilities would not provide us with good performance under such a scenario. However, Hibernate was shown in our tests to perform similarly to vanilla JDBC on the large object graphs produced by our system. We thus gain the ability to let our services operate on Java objects and know that we are not suffering a performance penalty by delegating the generation of SQL statements to Hibernate.

To increase performance, we make use of Hibernate Search (which in turn delegates to Lucene) to create and maintain an index into the data model of our services. Indeed, Hibernate Search allows us to build the index in a concurrent fashion and update it alongside inserts and updates of the underlying database. This feature allows the index to be used to retrieve objects directly after they have been created without the need for a periodic background indexing process. The use of Lucene underneath Hibernate Search provides additional benefits because we can access Hibernate Search's APIs (backed by Lucene) directly for those services that only perform reads on the database. This is because Lucene's projection ability allows it to serve as a highly-scalable map storage solution providing access to fully-hydrated Java objects without accessing the underlying database.[2]

2.2 The Infrastructure's Layered Service Model

Applications built on top of our infrastructure have access to a layered set of services and APIs that mimic the areas listed on the left-hand side of Fig. 1. These layers are *domain*, *persistence*, *service* and *application*. Each layer provides services that make it straightforward both to leverage the existing capabilities of the infrastructure and to add new functional capabilities. We discuss each layer in turn.

2.2.1 Domain

Our domain layer consists of services for defining the domain objects that are important to the higher-level services and applications in the infrastructure. For crisis informatics research—and with our current focus on Twitter—our domain objects consist of concepts such as User, Tweet, Community, etc. As mentioned above, all of the Java classes that represent our domain objects are properly annotated to make use of services provided by JPA and Lucene. It is straightforward to include these existing domain objects in new services and applications by simply adding the fully-qualified Java class names of the desired objects in the new application's *persistence.xml* file (a configuration file used by JPA). This inclusion based on how it is configured will cause JPA to either generate a new database that includes all of the required table and column definitions for the included domain objects or to gain access to a pre-existing database with those tables to allow the application to share access to these objects with other applications and services. The latter is what we did when deploying the Twitter social graph service; it simply makes use of the User and Tweet objects already created by the data collection service and adds the tables for its Community object alongside them.

This approach of using JPA ensures that all applications that interact with our infrastructure are correctly describing the same domain objects. This feature becomes incredibly important when you start to create multiple, highly-concurrent applications that access, update and manipulate the same domain objects to ensure that data collection and analysis can occur without corrupting the data store or duplicating information across services that happened to describe similar domain objects in different ways.

2.2.2 Persistence

In a complex application domain—such as mass emergency response—where data collection and processing must occur concurrently to enable real-time data analytics, controlling how active services can persist their information and dealing with conflicts when they arise is a difficult task. All of our infrastructure's persistence logic (including its query framework) lives in our persistence layer. This layer offers services that encapsulate all the knowledge required to persist and retrieve objects from the database. Indeed, the services in this layer are configured with a set of sensible defaults that allow most applications to simply adopt them and get access to existing data stores, or have the ability to create their own data stores with little to no configuration.

It is in this layer that we placed the mechanisms for making use of the customized transaction callback that we discussed in Sec. 2.1. Our hybrid transaction framework makes defining transactions easier than programmatic frameworks (via our use of Spring transactions) but offers more control than a declarative transaction style (via our transaction callback) allowing for fine-grain specification of transaction boundaries. This hybrid framework is then leveraged by all of the higher-level services and applications to ensure maximum performance while running concurrently while also maintaining data consistency. What is essential to making this work is avoiding the default use patterns of JPA entity managers that are encouraged by Spring's framework documentation and countless tutorials around the Web.

These tutorials lead software engineers down the wrong path by encouraging a pattern that creates an entity manager and holds it open for the entire length of a web application's request-response cycle. For long-running concurrent computations, this pattern can cause a large graph of stale objects to be maintained by the entity manager which will result in a failure to persist on flush due to shared objects being modified and persisted by shorter running computations.

[2]Hibernate is used to "hydrate" Java objects from values stored in a database. If an object does not change after it has been inserted into the database, Lucene can cache the hydrated object and return it without accessing the database.

Our transaction framework works with the entity manager to ensure that shared objects are held for the smallest amount of time possible and that changes are properly flushed on the application-defined transaction boundaries.

We ourselves had been led down the wrong path by the recommended use of entity managers and found that our social graph service would prevent the data collection service from doing its job when the social graph service was mapping a large Twitter community. The social graph service was holding on to too many shared objects (Twitter users) for too long a time and the performance of the (critical) data collection service would grind to a halt. This situation led to our development of the hybrid transaction framework and a rejection of the "best practices" around entity managers to solve our performance problems.

2.2.3 Service

The service layer provides a home for domain-specific services that provide significant amounts of functionality to higher-level applications. For instance, our infrastructure provides the highly robust and concurrent RequestService in that layer; it hides all of the complexity of accessing Twitter's various APIs and even returns results as ready-to-use Java objects, hiding all of the effort that goes into binding Twitter's on-the-wire data formats by our custom data binders. All of our services make use of the same data binders; this allows for a single point of update when third-party services modify their object representations.

The services in the service layer heavily leverage the persistence layer and its transactional capabilities. This allows applications to access domain objects without having to write their own persistence code. Indeed, importing the services layer into an application provides its developer with access to any domain object stored by our infrastructure.

2.2.4 Application

Creating new applications on top of our infrastructure is straightforward. Our infrastructure provides a variety of APIs both via Spring MVC (in which web applications import classes from our service, persistence and domain layers) and REST-based web service endpoints that support anything from desktop applications, command line tools and even iPad/iPhone applications. As a result, we have command-line tools that can collect a given Twitter user's social graph and a given Twitter users's full tweet stream (while indexing the stream in Lucene at the same time). These command-line tools are thin shells that import the domain-specific services defined in our service layer and then string together the correct set of method calls to retrieve, process and store the data requested by the user that invoked them. We have web applications that collect tweets from Twitter's Search and Streaming APIs simultaneously, and we have an iPad application that can be used to search those tweets both during and after the collection. Since all of our applications make use of the same domain objects and the same high-level services, it is easy for them to work together, share data, and build on each other's capabilities.

3. FUTURE WORK

Our future work involves research and development on a number of fronts. We will continue to add domain-specific services to our service layer to allow data collection from other social media sites and to provide new analysis and visualization capabilities. We will be exploring ways to increase the scalability of our infrastructure by integrating a distributed object cache that will allow us to replicate our database across a number of machines. Finally, we are actively working on ways in which we can integrate distributed computing frameworks such as Hadoop and Cassandra into our infrastructure so that massively-parallel computations can be applied to our large datasets to enable real-time data analytics for crisis informatics investigations.

4. CONCLUSIONS

We have reported on the design and implementation of a large-scale data collection and analytics framework that serves to support multidisciplinary research in the area of crisis informatics. The importance of the research and application domain has placed significant demands on the quality and nature of the software tools developed to support this research effort, with expectations on reliability and scalability far exceeding what is typical in software engineering research contexts. We have demonstrated that production-class software frameworks can be used to meet these demands, but the associated effort is non-trivial and requires research to develop customizations when the constraints of the application domain force developers away from the traditional ways of using these frameworks.

Space constraints prevent us from going into detail on our software development processes, but we have also found the need to incorporate production-class development techniques into our research environment. In particular, our current process makes use of a distributed version control system that is integrated with a continuous build system to ensure that new code does not break existing functionality. In addition, we make use of an automated dependency management system that ensures that all developers are using the same set of infrastructure components and libraries.

We believe our infrastructure represents the bare minimum of capabilities required to perform useful crisis informatics research and we believe our experience in having to raise both the quality and reliability of software prototypes in multidisciplinary research settings is an important lesson for the software engineering community.

5. ACKNOWLEDGMENTS

This material is based upon work sponsored by the NSF under Grant IIS-0910586.

6. REFERENCES

[1] 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 *2010 BCS Conference on Visions of Computer Science*, Apr. 2010. Article 8, 12 pages.

[2] L. Palen, J. Martin, K. Anderson, and D. Sicker. Widescale computer-mediated communication in crisis response: Roles, trust & accuracy in the social distribution of information. `http://www.nsf.gov/awardsearch/showAward.do?AwardNumber=0910586`, Sept. 2009.

[3] L. Palen, S. Vieweg, and K. M. Anderson. Supporting 'everyday analysts' in safety- and time-critical situations. *The Information Society Journal*, 27, 2011.