Architectural Implications of Social Media Analytics in Support of Crisis Informatics Research

Kenneth M. Anderson, Aaron Schram, Ali Alzabarah, Leysia Palen
Project EPIC
Department of Computer Science
University of Colorado Boulder
{ken.anderson,aaron.schram,ali.alzabarah,leysia.palen}@colorado.edu

Abstract

Crisis informatics is a field of research that investigates the use of computer-mediated communication—including social media—by members of the public and other entities during times of mass emergency. Supporting this type of research is challenging because large amounts of ephemeral event data can be generated very quickly and so must then be just as rapidly captured. Such data sets are challenging to analyze because of their heterogeneity and size. We have been designing, developing, and deploying software infrastructure to enable the large-scale collection and analysis of social media data during crisis events. We report on the challenges encountered when working in this space, the desired characteristics of such infrastructure, and the techniques, technology, and architectures that have been most useful in providing both scalability and flexibility. We also discuss the types of analytics this infrastructure supports and implications for future crisis informatics research.

1 Introduction

The field of crisis informatics has arisen as a topic of study starting in the mid 2000s in response to the pervasive access and use of technology by members of the public during times of mass emergency. First coined by Christine Hagar during the UK foot and mouth disease crisis [3,4] and then expanded by the work of Leysia Palen [6, 7, 9, 10], crisis informatics seeks to understand the range of techniques, services, and technology that can be brought to bear to better understand how the public uses technology to respond to disasters and mass emergencies. It also studies how that “informal” response interacts with and influences the “formal” response by government agencies. From that understanding, crisis informatics hopes to guide the design of future technology to better serve those needs with an eye towards making society’s response to disaster more resilient and effective.

In our work as part of Project EPIC [8], we have investigated various software architectures and software components needed to enable crisis informatics research [1,2]. Our approach is to fundamentally support the larger international endeavor of crisis informatics that is geared towards developing end-user tools. These tools are typically referred to as *crisis dashboards* or *crisis mash-ups* and they attempt to display information about a specific event in various ways including reports, live streams, and maps.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

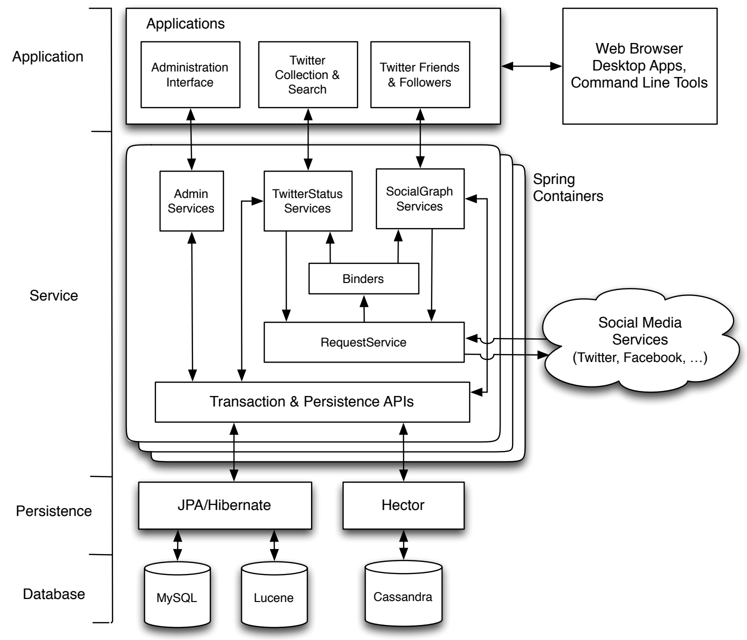

Figure 1: The Project EPIC Architecture for Scalable and Reliable Data Collection and Analysis

However, our larger goal has been to go further than individual tools and create *crisis informatics infrastructure*: a platform that can be used to develop general software tools that can be used on a variety of events and that can enable the longitudinal study of hundreds of similar crisis events over many years. With that goal in mind, we have designed and developed a layered software architecture with clear delineations between physical storage, persistence-related services, domain-specific services, and applications that can be implemented by a flexible, highly available, horizontally scalable, reliable, and robust software infrastructure [1].

Within this framework, we have investigated the capabilities of various production-class software components (such as Cassandra, Lucene, MongoDB, MySQL, and Spring) and developed the software that glues them together into a reliable and robust system that can collect large amounts of social media data during times of mass emergency while maintaining 24/7 operation with 99% system uptime [2].

Our current infrastructure adopts the logical software architecture shown in Fig. 1 and is deployed on a cluster of machines that includes one machine for web-based applications, one for analytics, one for backup, and four nodes for the storage of terabytes of collected social media data. With this infrastructure, Project EPIC has collected nearly three billion tweets across hundreds of events over the past three years.

Below, we highlight the software engineering challenges encountered when working to capture and analyze the data needed to support crisis informatics research. We present the types of questions asked of the social media data sets generated during a crisis event and the range of problems that such data sets present for collection. We also discuss the adjustments we have made to our crisis informatics infrastructure to support social media analytics and provide insight into the types of analysis enabled by our software platform.

2 Analytics for Crisis Informatics Research

The software infrastructure in Fig. 1 is able to collect millions of tweets per day for many weeks to track the social media conversations about crisis events during their immediate response and recovery period and sometimes during the warning period, depending on the nature of the hazard. Once the conversation diminishes, collection for events can end and analysis can begin. Crisis informatics researchers may then pose a wide range of queries about collected data sets [5] including identifying how many unique users contributed to the sets; the most retweeted tweets; tweets containing popular links; the most influential twitter users; the percentage of tweets that were retweets; number of geolocated tweets; a high-level description of common terms and their evolution; etc. Other questions include more prosaic metrics such as what search terms were used to generate the data set, the start and end dates of the collection, volume of tweets per day, and so on.

The analysts also like to identify a smaller representative set of tweets to study the event in more qualitative detail. They seek to understand how people use social media to cordinate or collaborate during events with each other or with emergency response organizations. The challenge is that even these data sets can consist of millions of tweets and a group of human researchers can only perform in-depth analysis on a set of a few thousand tweets. As a result, an additional form of analysis is required to identify the rules and procedures for creating a representative set of tweets. These rules and procedures can be different across event types, but in general will include the need to filter out certain types of tweets ("filter out tweets that contain the word 'pray' ") or to sample tweets based on various characteristics ("include at least one tweet from all users who contributed to the entire set") or metrics ("include only tweets that were retweeted at least 100 times") [5].

3 Challenges in Analyzing Twitter Data

From a software engineering perspective, there are numerous challenges that must be confronted when both collecting and analyzing large sets of Twitter data. On the collection side, the Twitter Streaming API can deliver large amounts of tweets in real time but cannot provide access to tweets generated in the past, while the Twitter Search API provides access to past tweets but significantly limits that access to just a small sample, sometimes going back weeks but more typically just a few hours. Furthermore, most Twitter API methods are rate limited which constrains the speed at which a data set can be generated. For instance, the API method that allows the retrieval of the last 3200 tweets generated by a user is restricted to 720 requests per hour where each request can retrieve 200 tweets of a particular user. If you assume that nothing goes wrong and all users have at least 3200 tweets, then these rates limit you to the collection of just 45 users per hour. The actual average is higher than that (since not all users have 3200 tweets) but not significantly and problems can and do occur over the long periods of time that it takes to do this type of collection. Such problems include network disconnects, hardware failures, Twitter service failure, and more. All of these problems can be handled—but not easily—and the solution requires significant engineering effort, system administation skill, and the use of (potentially unfamiliar) concurrency, distributed systems, and NoSQL techniques.

On the analysis side, further complications abound. Twitter data can be "messy" and a significant amount of time has to be invested to get it into a state where it can be effectively analyzed. In particular, each tweet can contain a different set of fields (hindering the ability to write generic processing code) and can contain text in multiple encodings (hindering the ability to parse and display them). In addition, an individual tweet does not contain all of the information that an analyst might want. For instance, the user metadata of a tweet indicates how many "followers" that user has but not who those followers are. If an analyst wants to pull follower information for the Twitter users of a data set that requires a second, significant data collection effort, subject once again to the rate limits and problems discussed above. In addition, the collection of follower graphs is a signficant engineering task itself. A collection of just "two hops" of follower/following relationships can quickly lead to millions of user objects that need to be collected, stored, and later analyzed.

Another issue that occurs with the analysis of tweets is that the values of tweet metadata (e.g., retweet count, follower count, favorite count, etc.) is dependent on when the tweet is collected. If a tweet is collected during an event, the values reflect what had happened to that particular tweet at that particular time. Thus a tweet that was just generated will have a retweet count of zero and may sit in a data set classified as "uninteresting" when it later goes on to be retweeted thousands of times. Users who had only ten followers at the time they contributed their first tweet to a data set may go on to have thousands of followers if they generate interesting content during that event. However, if that first tweet was the only one that matched the search terms being used to collect the event, subsequent analysis may classify that user as "uninteresting" since at the time his follower count was low.

One way around this problem is to collect again all tweets that make up a data set once the initial period of interest for that event is ended. This would ensure that all tweets in the data set are stored with the latest set of metadata. This approach would help reveal all tweets and users that went on to become "popular," but this approach is impractical when the size of the initial data set is large (100s of GBs). Firstly, the original metadata values would be lost or, if both versions of the tweet are kept, the size of the data set doubles. Secondly, one is required to switch from collecting tweets via the high-velocity Streaming API to the rate-limited REST API, which, for large data sets, means facing months of data collection.

Fortunately, for long-running collection events, these problems are mitigated. Interesting users gain influence by tweeting lots of useful information. Each tweet they contribute to the data set contains updated values of their followers count. A useful tweet that is retweeted multiple times is "embedded" in the retweet; the embedded tweet then contains updated values for retweet count, follower count, etc. As a result, we have developed analytical tools that track the evolution of metadata values for tweets and Twitter users over the time of a collection. Once the set of truly popular tweets and Twitter users for an event has been identified, a smaller, more focused data collection can occur via the REST API to ensure that we have the most recent metadata values for popular tweets and the complete set of tweets generated by influential users.

4 Extending the Infrastructure to Handle Analytics

The Project EPIC infrastructure, initially designed to solve the problem of collecting large amounts of social media data in a scalable and flexible fashion [1, 2], is well situated for extension to handle the challenges of social media analytics. New data stores, services, and applications can be added to the relevant layers of the architecture to provide analytical capabilities. We currently make use of four different approaches to performing analytics over our social media data sets.

Our first approach to analytics is enabled by exporting the tweets collected for an event from Cassandra to the file system of our analytics machine. Once exported, scripts written in Ruby, Python, or Java can process this data to look for the answers posed by Project EPIC analysts. These scripts are written in a highly-iterative fashion on subsets of data to demonstrate the feasiblity of answering a question or to test out the effectiveness of a particular algorithm in obtaining the answer as efficiently as possible. Once this experimental stage is complete, these algorithms are encapsulated in MapReduce jobs and applied to the entire data set.

Our second approach to analytics involves migrating event data out of Cassandra and into a NoSQL store that is suited more for analytics; while Cassandra shines as a reliable and scalable data store, it is less suited for more open-ended analytics where queries are not known ahead of time. We currently make use of MongoDB and Hadoop/HDFS for this purpose. We have written software that can export events out of Cassandra and import them into MongoDB or HDFS. We have standard queries (as described above) for these data sets written as MapReduce jobs in both MongoDB and Hadoop that will extract the information that our analysts require as a baseline. These queries currently run on a cluster of four machines generating the answers to queries on data sets with tens of millions of tweets typically in just a few hours. As both MongoDB and Hadoop are "horizontally scalable," Project EPIC is in a situation where if we need to process larger data sets (100M+ tweets) in a similar amount of time, we simply need to acquire new machines and add them to the existing cluster.

Our third approach to analytics is looking at technologies that can enable "real time analytics." For Project EPIC, real-time analytics means producing the answers to our standard queries on data sets that are still being collected. This capability would be useful both in the initial onset of an event to help our analysts determing the keywords that we should be using to collect a representative set of tweets for the event and during the event itself to highlight influential users and tweets that are worthy of additional study after the event. We have three strategies that we are pursuing in this area, all in the beginning stages of development. These approaches are 1) making use of commercial tools, 2) making use of stream-processing frameworks, and 3) making use of periodic MapReduce jobs. The first strategy is to make use of commerical software designed for the processing of time series data. We are currently using Splunk to provide a "dashboard" for monitoring the volume of tweets of an event and to make initial queries on the types of conversations that are occuring for an event.[1] We do not store all of the tweets for an event in Splunk. Rather we delete older tweets (where the definition of "old" evolves over the course of the event) to make way for more recent tweets, allowing the dashboard to present a current picture of the event to our analysts.

The second strategy is to investigate the use of stream processing frameworks, such as Storm, to generate answers to our standard questions on partially-collected data sets. Storm provides us with the ability to generate metrics as tweets arrive, before they are stored in Cassandra. These metrics (number of unique users, number of tweets per hour, etc.) can be stored in a small relational database and displayed using a web-based dashboard.

The third strategy is to make use of periodic MapReduce jobs that run on a partially collected event and report back the answers to our standard set of questions. Making use of MapReduce jobs in this fashion provides the ability to be more expressive in the types of manipulations performed on the data set, and can allow for the production of more complicated metrics, such as the number of unique users contributing tweets matching an event's keywords per hour. We are investigating whether it is more useful to run these jobs on the entire data set (which implies that performance will be less and less "real time" as the data set grows) or on a "sliding window" of tweets (e.g. the last 50,000 tweets collected) that can expand or shrink as needed to enable fast query execution. In the future, we will be looking at how technologies such as Pig and Spark enable the creation of MapReduce jobs at a higher level of abstraction. Such technologies often integrate with existing NoSQL stores allowing us to avoid migrating data and instead execute queries directly against our Cassandra data sets.

Finally, our fourth approach to analyzing our large data sets is via the use of graph databases (e.g. Neo4J, FlockDB, and Titan). While the collection of social graphs on Twitter represents a significant engineering challenge, we have software that can collect these graphs based on a given starting point (i.e. Twitter user). Now we are developing software that makes use of a graph database to read in graphs to allow our analysts to traverse the network of followers surrounding the influential contributors of a particular event with an eye towards visualizing and understanding the growth of these networks over the course of an event. We are also interested in incorporating non-Twitter data into this database where some nodes represent URLs that appear in tweets to determine if it is possible to track the flow of information about an event in and out of Twitter.

In summary, we view all four of these approaches as critical to providing a comprehensive analytics platform for crisis informatics research. To give a sense for the utility of one of these techniques, we now present a more in-depth look at the types of analysis that our work with MongoDB is providing.

5 Challenges Using MongoDB for Social Media Analysis

Project EPIC makes use of Twitter to track the social media conversation about large-scale crisis events. Members of the public increasingly make use of Twitter during mass emergencies [7]. Some of the tweets can contribute data to build "situational awareness" of the emerging event. Others have URLs to information that resides in other forums. As a result, Twitter often serves as a useful map to the range of online communications that occurred around a mass emergency event. Project EPIC makes use of Twitter's Streaming API to collect tweets; our infrastructure provides a web-based application to create/modify events, add/disable

[1]Citations to popular software tools and frameworks—e.g. Splunk, Storm, and Cassandra—are ellided as information about these projects are easily discovered via internet-based search tools.

keywords associated with an event, and manage when and for how long events are submitted to Twitter for collection. The Streaming API delivers tweets in JSON format and a varient of JSON (known as BSON) is the native format used by a popular NoSQL data store known as MongoDB. As MongoDB is designed to support arbitrary queries over objects stored in this format, MongoDB has interesting potential for supporting the analysis of large sets of tweets. MongoDB is not, however, a panacea. The use of indexes and queries on a single instance of a MongoDB server quickly runs into problems that can only be solved via the use of more complex techniques and client-server configurations.

For instance, answering the “unique number of users contributing to a data set” question identified in Sec. 2 can easily be answered in MongoDB by first importing the data set into a MongoDB collection (e.g. `tweets`) and issuing a command similar to `db.tweets.distinct("user.id_str")`.[2] This command causes the MongoDB server to look at all documents in the `tweets` collection for an attribute called “id_str” stored in an “embedded document” labelled “user.” It keeps track of all the unique values found for this attribute, stores those values in an array, and, ultimately, returns that array to the caller for display and further analysis. This command executes quickly on small datasets, especially if an index has been created on the “user.id_str” attribute. However, on large data sets with many unique values, this command can encounter an internal limit of the MongoDB server which prevents the results of in-memory queries growing larger than 16MB.

For one of our smaller data sets—7.3GB of Twitter data consisting of 2.2M tweets with 932K unique users—we hit exactly this limit and were thus unable to use the `distinct()` command to calculate this particular metric. Instead, we had to make use of MongoDB’s ability to run MapReduce jobs to retrieve the answer. This is straightforward to do but does represent an increase in complexity. An engineer must switch from running the simple command shown above to code similar to the following:

```ruby
def map_command
  "function() { emit(this.user.id_str, 1); }"
end

def reduce_command
  "function(key, values) { return Array.sum(values); }"
end

mongo = MongoClient.new

db = mongo.db("norway")['tweets']

db.map_reduce(map_command,reduce_command,{:out => 'users',:verbose => true})
```

In particular, our map command is passed each input document in turn and generates an output document containing the value of the input document’s `user.id_str` attribute as a key and the number one as a value, i.e., {"key" : "1234", "value" : 1}. (Note: the input document is “passed” by setting the value of `this` to point at each input document in turn.) After the map phase is complete, MongoDB combines documents that have the same key by adding their values to an array. Conceptually, this produces documents that look like this: {"key" : "1234", "value" : [1, 1, 1, 1, 1, 1]}. These combinations get passed to the reduce function which in turn produces documents that look like this: {"key" : "1234", "value" : 6}. When the reduce phase has processed all combination documents, these final output documents are stored in a `users` collection in which there is one document for each unique user in the `tweets` collection. The `value` attribute for each of these documents stores the total number of tweets that user contributed to the data set. It is still

[2]The use of `id_str` over `screen_name` is preferrable for determining unique users in a Twitter data set. Users can (and do) change the value of the `screen_name` attribute whenever they want; the `id_str` value stays constant for the lifetime of a Twitter user account.

possible to encounter out-of-memory errors using this approach if you fail to specify an output collection. In that case, MongoDB keeps all combination documents and all final output documents in memory and will return the final set of output documents to the caller when the reduce phase is done. If this in-memory set of documents consumes more than 32MB of space, the call to `map_reduce` will fail. Instead, you must tell MongoDB to store the intermediate documents in a collection (as was done above on line 13). This causes MongoDB to store the intermediate documents on disk rather than in memory and with those conditions in place it becomes possible to reliably generate the answer to the "unique users" question independent of the size of the overall data set.

However, without some additional work, the time it takes to generate the answer to this simple query can still take a long time. The default configuration for MongoDB is for it to run as a single-threaded server instance. In such a configuration, the MapReduce job above will look at all documents in a collection during the map phase sequentially and will then process all the combination documents during the reduce phase sequentially using a single thread (even on a server with multiple processors/cores). To take advantage of MapReduce's ability to run in parallel on very large data sets, MongoDB must be configured to run multiple instances of the MongoDB database on multiple servers. It must be configured to "shard" the original database across those servers. The good news here is that the programmatic interface to MongoDB does not change in this set-up, but the administration of this configuration is non-trivial and one must deal with issues such as selecting the key used to partition the database across the shards. However, the benefits of this approach is horizontal scalability in terms of performance and disk space. If one needs additional storage space or one needs queries to run faster for a given data set, adding another server to the configuration will trigger automatic repartitioning of the database allowing for increased scalability and parallelism.

Once properly configured, MongoDB can offer powerful analytical tools on large data sets via disk-based MapReduce jobs like the one above. Two analysis features that we make use of in our more complex MapReduce jobs are MongoDB's geospatial indexes and full-text indexes. For the latter, we create full-text indexes on the text of a tweet. The full-text index allow us to search data sets with more complex queries such as, "all tweets that contain 'haiti' but not 'pray'." This functionality, in turn, provides our analysts with an ability to be more expressive in their queries and to broaden their research questions. We make use of MongoDB's geospatial indexes to help narrow down tweets in a given data set to just those users who were generating tweets local to an event. For instance, our 132 GB Hurricane Sandy data set contains 26M tweets generated by 8.2M users from all over the world. However, for one of our research questions, our analysts wanted to examine only tweets generated by users who were located on the eastern seaboard of the United States right before the landfall of the hurricane. After importing the data set and ensuring that the geocoordinates of the tweets matched the format expected by MongoDB, we generated a geospatial index and wrote a MapReduce job that searched for tweets that fell within a particular bounding box. The query took 13 minutes to run on our cluster and located the 101K users who were directly in the hurricane's path. This result was immediately useful as we were able to launch a new data collection in which all of the tweets ever generated by those 101K users were collected and stored for later analysis (looking, for instance, to see if these users had adopted Twitter because of Hurricane Sandy).

6 Conclusions

In this paper, we have highlighted the challenges associated with collecting and analyzing social media data. While our experience is related to the support of crisis informatics research, many of these issues are independent of application domain and our approach, techniques, software architecture—and its implementation as Project EPIC's software infrastructure—are broadly applicable to the collection and analysis of a wide variety of application domains and types of social media. We have learned valuable lessons, including insight into the questions most important to crisis informatics researchers, the challenges associated with collecting representative Twitter data sets, and the need for multiple approaches to analysis. A single analysis technique is insufficient to answer the wide range of questions posed across the different timescales in which those answers are needed. We also learned that it is critical to develop a reasoned, modular, and extensible approach to the design and development of social media analysis tools. It is otherwise impossible to perform this analysis at scale with sufficient functionality, flexibility, and performance to make meaningful progress on societal-scale issues.

One important implication of our experience is how work in this area requires a large team with a diverse set of skills and expertise to do this type of research effectively. Project EPIC required a team with skills not only in software architecture, software design, software engineering, web engineering, distributed systems, networking, and system administration, but also with skills in human-centered computing, information visualization, statistics, and qualitative/quantitative experimental methods. The former were needed to create, deploy, and operate the scalable, flexible, reliable, and robust Project EPIC software infrastructure but the latter were needed to identify the problems it solves in the first place, what questions it must answer, how it should present those answers, and how it supports the ongoing, highly-iterative, multi-method techniques required by crisis informatics research. With such a team, our interdisciplinary approach to the collection and analysis of social media has enabled significant progress to be made in the area of crisis informatics research.

Acknowledgment

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

References

[1] Anderson, K., and Schram, A. (2011). Design and Implementation of a Data Analytics Infrastructure in Support of Crisis Informatics Research: NIER Track. In *33rd International Conference on Software Engineering*, pp. 844–847.

[2] Anderson, K., and Schram, A. (2012). MySQL to NoSQL: Data Modeling Challenges in Supporting Scalability. In *ACM Conference on Systems, Programming, Languages and Applications: Software for Humanity*, pp. 191–202.

[3] Hagar, C., and C. Haythornthwaite. (2005). Crisis, Farming and Community. *Journal of Community Informatics*, 1(3):41âĂŞ-52.

[4] Hagar, C. (2009). The Information and Social Needs of Cumbrian Farmers during the UK 2001 Foot and Mouth Disease Outbreak and the Role of Information and Communication Technologies. In *The Socio-Cultural Impact of Foot and Mouth Disease in the UK in 2001: Experiences and Analyses*, eds. Döring, M. and Nerlich, B. pp. 186–199. Manchester University Press.

[5] McTaggart, C. (2012). Analysis and Implementation of Software Tools to Support Research in Crisis Informatics. MS Thesis. University of Colorado. 65 pages.

[6] Palen, L., and Liu, S. (2007). Citizen Communications in Crisis: Anticipating a Future of ICT-Supported Participation. In *ACM Conference on Human Factors in Computing Systems*, pp. 727–736.

[7] Palen, L., and Vieweg, S. (2008). The Emergence of Online Widescale Interaction in Unexpected Events: Assistance, Alliance and Retreat. In *ACM Computer Supported Cooperative Work*, pp. 117–126.

[8] Palen, L., Martin, J., Anderson, K., and Sicker, D. (2009). 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`.

[9] Palen, L., Anderson, K., Mark, G., Martin, J., Sicker, D., Palmer, M., and Grunwald, D. (2010). A Vision for Technology-Mediated Support for Public Participation & Assistance in Mass Emergencies & Disasters. In *ACM and British Computing SocietyâĂŹs 2010 Visions of Computer Science*, 10 pages.

[10] Palen, L, Vieweg, S., and Anderson, K. (2011). Supporting “Everyday Analysts” in Safety- and Time-Critical Situations. *The Information Society Journal*, 27(1): 52–62.