A Software Architecture for Progressive Scanning of On-line Communities

Roberto Baldoni, Fabrizio d'Amore, Massimo Mecella, Daniele Ucci
Cyber Intelligence and Information Security Center
Dipartimento di Ingegneria Informatica, Automatica e Gestionale
Sapienza Università di Roma, Italy
Email: {baldoni|damore|mecella}@cis.uniroma1.it , ucci.1208859@studenti.uniroma1.it

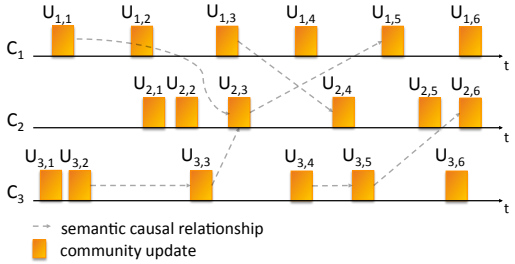
***Abstract*—We consider a set of on-line communities (e.g., news, blogs, Google groups, Web sites, etc.). The content of a community is continuously updated by users and such updates can be seen by users of other communities. Thus, when creating an update, a user could be influenced by one or more updates creating a semantic causal relationship among updates. This transitively will allow to trace how an information flows across communities. The paper presents a software architecture that progressively scan a set of on-line communities in order to detect such semantic causal relationships. The architecture includes a crawler, a large scale storage, a distributed indexing system and a mining system. The paper mainly focuses on crawling and indexing.**

Keywords*—*On-line communities, progressive scanning, MapR, Nutch

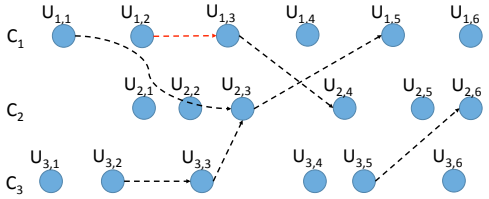
I. Introduction

On-line communities are nowadays a fundamental source of information in business and information security intelligence. Blogs, wikis, forums, Facebook specific pages, Twitter threads, Google+ sparks, Meetups, etc. contain information that can be used, if appropriately crawled and mined, for inferring trends and evolution about specific topics. In particular, a social community, and what it publishes, often influences other communities, and vice-versa, thus creating a network of causal relationships that can contain useful information about the evolution of a specific phenomenon.

To make this more concrete, let us consider a set $\mathcal{C} = \{C_1 \dots C_n\}$ of social communities, as shown in Figure 1(a). Each community C_i performs several updates $U_{i,j}, j = 1..m_i$ of the published information, which in turn influences the update(s) of some other communities. As an example, in the figure, it is shown how the update $U_{1,1}$ of the community C_1 has influenced the (published information) update $U_{2,3}$ of the community C_2; notably this update is also influenced by the update $U_{3,3}$ performed by the community C_3 and, in turn, influences the update $U_{1,5}$ of the same community U_1. The reader can observe how, in the meanwhile, community U_1 has performed several updates (cf. updates $U_{1,2} \dots U_{1,4}$).



(a) Evolution along the time of a set of social communities showing how an update is causally influenced by other updates



(b) Example of semantic causal relationships graph extracted from the execution of Figure 1.a

Fig. 1. Social communities and corresponding graph showing the influence among updates

A system supporting business/security intelligence among on-line communities should be able to produce a partial order as shown in Figure 1(b); here we can see that the system correctly captures all the causal relationships, but *(i)* the relationship $U_{3,4} \rightarrow U_{3,5}$ is not captured (*false negative*), and *(ii)* the relationship $U_{1,2} \rightarrow U_{1,3}$ is considered, which indeed is not valid (*false positive*).

The aim of this paper is to present the architecture of a software system able to detect influence among updates of a set of communities. The system should progressively scan a set of on-line communities (selected on the basis of some criteria that is out of the scope of our work) in order to infer causal relationships among updates of the published information; the correctness of the system is measurable in terms of *precision* and *recall*, as customary in information retrieval and mining literature.

Interestingly the notion of causal relationships in message passing distributed systems is a well known concept developed in the late seventies by Lamport [12], and then used in computer networks to reduce the network non-determinism at receiver side due to unpredictable message delays [2], [16]. In the context of communities , we are interested in capturing causal relationships among updates looking at the fact that the authors of the text of the update $U_{i,j}$ have been causally affected during the writing by the text of an update $U_{k,h}$. As a matter of fact, if the author of $U_{i,j}$ is influenced by $U_{k,h}$, then there is a causal dependence in the Lamport sense between the publishing of $U_{i,j}$ and he one of $U_{k,h}$. The viceversa is not necessarily true, also if $U_{i,j}$ and $U_{k,h}$ have been published on the same community, as shown by the graph in Figure1(b).

The following of this paper is as it follows. In Section II the software architecture is introduced. Section III presents the software technologies used to implement the proposed architecture, whereas Section IV describes the interactions among the different components. Section V provides details on the web crawling and indexing layers. Section VI discusses relevant works, and finally Section VII concludes the paper by outlining future research activities.

II. Software Architecture

The proposed software architecture needed to perform progressive scanning of on-line communities is mainly composed by three different layers, each one in charge of executing specific tasks. The functionalities of the different layers, shown in Figure 2, can be summarized in the following ones:

- *web crawling and indexing* to progressively scan the updates in on-line communities;
- *storage and processing*, where to efficiently store the data and metadata coming from the crawling and indexing phases;
- *data mining*, where to detect the semantic causal relationships among updates along the line of what described in the introduction.

As shown in Figure 2, there is a strong interconnections between the different layers, therefore, as further clarified in Section III, all the technologies employed for the proposed architecture are highly integrated with each other. Figure 2 also highlights

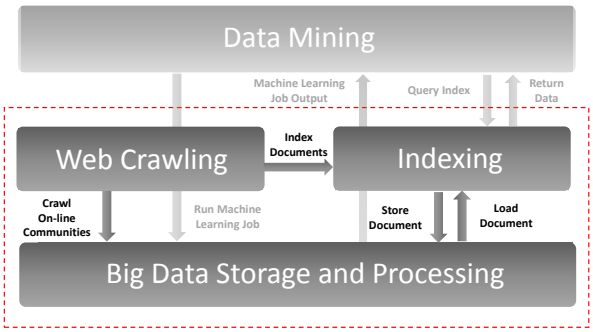


Fig. 2. Proposed software architecture for progressive scanning of on-line communities. The red dotted square contains the layers whose functionalities have been already implemented

which are the layer functionalities implemented at time of writing: Big Data storage and processing and web crawling and indexing. Data mining layer functionalities have been designed but not yet implemented.

In Section III, the different frameworks chosen to implement the proposed architecture will be presented and briefly discussed.

III. Software Architecture Implementation

In order to develop the architecture described in Section II, the software products used are: MapR, Apache Nutch, Apache Solr and MapR Mahout.

The storage and processing layer is managed by both MapR and Solr. MapR [13] is an industry-standard Apache Hadoop-derived framework easy to use, dependable and especially fast. Being derived by Hadoop, MapR includes a MapReduce module for parallel processing of large data sets. The key improvements introduced by MapR concern distributed file system's performance and robustness, allowing continuous read-write accesses. Moreover, it is Hadoop-API compatible and supports the Hadoop distributed file system abstraction interface. These latter features are particularly important to maintain compatibility with the Hadoop ecosystem and with the other Hadoop-related projects. Instead, Solr [19] is an open source enterprise distributed search platform highly reliable, scalable and fault tolerant. These capabilities are handled by a Solr sub-layer, called SolrCloud [21]. In this way, Solr is able to provide distributed indexing and searching capabilities, including full-text search, hit highlighting and faceted search.

The upper software layer comprises the Apache Nutch [20] web spider and, again, Solr. Once Nutch has crawled the web pages/documents of on-line communities, these are indexed to perform searches upon them very quickly. Actually, Nutch and Solr do not interact directly, but rather they communicate using

the underlying layer. More details on this will be addressed in the next section. As previously mentioned, Apache Nutch is a highly extensible, robust and scalable open source crawler supporting the MapReduce paradigm. Furthermore, Nutch observes politeness and implements a robot-exclusion protocol. They guarantee, respectively, that a web site is not overloaded with requests from multiple fetchers in rapid succession and site owners be able to control which parts of their site may be crawled.

The software architecture completes with the data mining layer , which relies on Apache Mahout. Mahout [23] is a library useful to produce free implementations of scalable machine learning and data mining algorithms, through which is possible to establish correlation between documents and apply mining techniques. Many of these implementations exploits the MapReduce framework and, therefore are suitable to be run on a MapR cluster.

IV. Software Modules Interactions

In the previous section, all components have been presented. As depicted in Figure 3, each layer interfaces with one or more other ones . Starting from Nutch, all the interactions between modules will be now discussed.

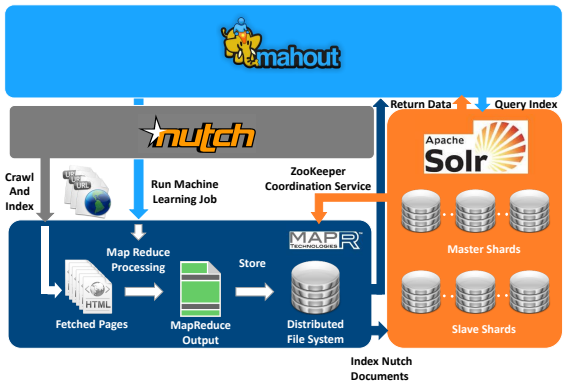


Fig. 3. Existing interactions between different software modules of the proposed software architecture.

The Nutch web crawler makes available a *jar* job which can be spread over a cluster using the MapR layer. In this way, Nutch is able to benefit of an improved distributed file system and of a MapReduce processing style. Passing in input a set of URLs to crawl and a list of Solr servers to the underlying layer, the crawling and indexing jobs can be run in distributed mode.

MapR provides large-scale processing through MapReduce, BigData storage, ZooKeeper coordination and synchronization services. Once MapR receives the Nutch's input, it distributes the crawling and, subsequently, the indexing jobs over the cluster. After a preliminary phase in which the Nutch's crawling database, presented in Section V, is populated with the data received in input, Nutch uses its internal components to obtain a single MapReduce structured output file, later stored in the MapR distributed file system. The MapReduce output file contains metadata, the actual HTML source-code and the parsed content of the previously fetched pages. Once the crawling task terminates, relying on the MapReduce framework, MapR starts preparing the pages-related data to be indexed, called NutchDocuments, and sends them to the Solr servers. Further information about how this documents are built are reported in Section V-A.

As mentioned in Section III, SolrCloud enables highly reliability, scalability and fault tolerance. This is realized through replication [22] and sharding [24] techniques. The sharding technique allows to split an index into multiple pieces, called shards. Each shard lives on its own server and possible incoming queries are sent to all the shard servers. Combine these two techniques is absolutely necessary to manage Big Data and queries from multiple sources. When the Solr layer receives the Nutch Documents, Solr indexes them to multiple master shard servers and replicates them on the correspondent slave shards. In order to manage distributed indexing management, automatic load-balancing and fail-over queries, Solr depends on the coordination and synchronization services offered by MapR's ZooKeeper. Finally, Solr is in charge to reply to index queries coming from the mining layer, the one that exploits the Apache Mahout software library.

Mahout exploits Solr and MapR to, respectively, retrieve input data on which apply machine learning and data mining algorithms and to execute large-scale processing using the MapReduce platform. From an abstract point of view, the output of this processing is the graph of Figure 1(b). This graph will be returned to the top layer (possibly a human operator to further investigate data correlation through appropriate additional software tools). Mahout can perform recommendation mining, document clustering and classification. Usually, for clustering and classifying documents, it is necessary to convert their raw text into term vectors which can be later consumed by the Mahout algorithms. In the proposed architecture, the documents are stored in a distributed Solr index which, properly configured, can provide for the use of term vectors. They allow for retrieving a single-document index for any document ID of the index itself. Therefore, given any document ID, Solr can quickly list all its unique terms in sorted order, and for every term it can rapidly know its original positions and offsets. Mahout can, hence, build its own term vectors starting from Solr ones and then apply clustering and classification algorithms to them.

V. Apache Nutch: Web Crawling and Indexing

Apache Nutch is the ideal candidate for web crawling and, at the same time, to be run on top of MapR. Being MapR

an Apache Hadoop-derived framework, it integrates perfectly with Nutch. In fact, initially Apache Hadoop was born as a Nutch sub-project to become, after few years, a stand-alone top-level project. This allowed a very easy interfacing between the MapR and Nutch software layers.

The Nutch's web crawling capabilities can be easily used for progressive scanning of on-line communities. The basic Nutch configuration already provides a desirable feature required for this project: the timestamp representing the time instant when a web page is fetched. The temporal aspects are also reflected in the web crawler's internal data structures. It maintains a:

- crawl database, in which are stored the information about every URL known to Nutch. This comprises metadata and page statuses;
- link database containing, for each known URL, the list of links, both the source URL and anchor text of the link;
- set of segments. Each segment is a set of URLs fetched as a unit.

Crawl database and segments, in particular, are responsible for keeping time-related information. Each time a set of URLs is due for fetch, they are grouped in a new segment that traces the date and the time in which it is created. Within the segment, also the metadata related to the actual page's fetch time is saved. The data related to the URLs to fetch are maintained in the crawl database.

Given this internal structure, it is straightforward to use Nutch to crawl iteratively the web to provide temporal snapshots of on-line communities. Hence, the basic configuration of Nutch helps, but it is not enough to guarantee progressive crawling. In order to do this, few adjustments are needed and they will be presented in the following.

Finally, thanks to its link database, it is possible to study the crawled web structure and apply mining techniques on this too. While web content mining mainly focuses on the inner-documents' structure, the web structure mining can capture relations at inter-document and web site level [11].

A. Configuring Nutch for on-line progressive scanning

The configuration issues do not depend only on the project's requirements, but also on the software pieces with which the crawler has to interact. Nutch is, in fact, just a web crawler and does not have any distributed indexing or searching capability. To make the content crawled by Nutch searchable, Solr is employed. As MapR and Nutch, Solr is in turn perfectly integrated with the latter. As already said in Section II, Solr can be run upon its own distributed infrastructure, called Solr-Cloud, giving to it the power to process data in a distributed manner, high availability and fault-tolerance. Solr relies on a XML schema to know which fields the document can contain and how those fields should be dealt with when adding documents to the index or when querying. In this work, Solr documents coincide with the web pages crawled by Nutch. Nutch has its own copy of the Solr XML schema and, in its configuration files, the mapping between the Solr fields and the ones that Nutch has to send to the distributed index. A correct indexing requires a perfect alignment between the two XML files.

The default schema provides for a unique ID, that is, the URL of the web page. If the on-line scanning is supposed to be progressive, such approach would fail immediately when the same URL is crawled twice, causing an overwriting of the previous snapshot of the same page. In order to avoid this, a new document ID has to be defined during the indexing phase. In Nutch, this can be accomplished by the use of plugins. They all define extension points with which is possible enhancing the Nutch's features. Through the implementation of a new extension for the `IndexingFilter` extension-point, a new ID has been constructed in the following way: the web page URL is concatenated with a version 4 Universally Unique IDentifier (UUID) and, then, the result of the concatenation is hashed using the SHA-256 cryptographic function. Its choice has been guided by two reasons: no collision has been found, even due to theoretical attacks, and the need of having a unique ID in the distributed index. Such ID is a hexadecimal string encoding the SHA-256 hash. Analogously, in the same way or using other extension points, a set of user-defined fields can be added to the Solr schema and, therefore, to the Solr index.

Another aspect to take into account is: progressive scanning needs a continuous crawling activity. A nice feature embedded in the Nutch crawler, but not enabled by the default, is the Adaptive Fetch Scheduling. This type of scheduler will adapt the page crawling frequency to the rhythm of page changes and set the next schedule time accordingly: every time a web page is fetched, the Adaptive Fetch Scheduler checks if it has changed. In the positive case, the next fetch interval would be decreased, otherwise increased up to a minimum or a maximum value, respectively. This implies that frequently updated web pages, eventually, will be fetched more than the other ones. Obviously, the `TextProfileSignature` has to be enabled to guarantee that the page will appear to be modified when its content has been actually updated and not in case of a simple http header modification, such as date refreshing.

VI. Related Works

In the last decade, thanks to the advent and to the development of new technologies, the amount of data exchanged and stored on the Internet is increased exponentially. The epochal change has been characterized by the transition from traditional data to Big Data. This term includes all the aspects related to information management and analysis technologies that exceed the capability of traditional data processing technologies

[7]. Big Data differs from traditional technologies in volume, velocity and variety. They mean, respectively, the amount of data, the rate of data generation and transmission and the types of structured/unstructured data [3]. According to IBM [8], human beings now create 2.5 quintillion bytes of data per day and their creation rate has increased so much that 90% of the data in the world today has been created in the last two years alone.

In this scenario, the need for new technologies, able to analyse massive data sets, has grown. The dominant and most popular technology for Big Data batch processing is Hadoop. Its effectiveness has been proven, lately, by the Zions Bancorporation's case study in which security and forensic jobs using Hadoop have been performed in near-real-time fashion [4]. These results have dramatically torn down the procedures' execution time of Security Information and Event Management (SIEM) tools, in which even simply loading the previous day's logs was a challenge taking up to one day. Taking advantage of Hadoop framework, in combination with distributed machine learning libraries, analysts can have at their disposal 3^{rd} generation intrusion detection systems able to perform deeper analytics on the data, providing a consolidated view of security-related information in near-real-time. In an analogous manner, this project plans to use the power of an Hadoop-derived product and an open-source framework for developing machine learning and data mining algorithms to correlate Big Data coming from a set of on-line communities. Such technologies can be also used for enterprise events analytics, NetFlow monitoring and against Advanced Persistent Threats (APT) [3]. Even if Hadoop is considered the leading technology in Big Data analytics, new trends are emerging [1]. These can be divided into paradigms going beyond MapReduce and paradigms based on it. The first ones, in particular, include real-time and incremental analytics such as Storm [18] and Dremel [14], respectively. Storm provides a distributed real-time computation system capable to process unbounded data streams, while Dremel analyzes read-only nested data by combining multi-level execution trees and columnar data layout. This allows it to run aggregation queries over trillion-row tables in seconds.

Regarding the middle layer of the proposed architecture, it has to cope with many challenges. These are mainly related to web crawler: it has to be scalable, robust to spider traps, efficient and polite. This latter one is a property that any crawler shoud respect: it should crawl only allowed pages and respect specifications from webmasters on what site portions can be crawled. A key aspect to consider is, furthemore, the crawling strategy adopted: in [15] a model and a taxonomy of crawl ordering techniques are widely presented. According to which is the chosen one, two different parameters can be affected: *coverage* and *freshness*. The first represents the fraction of desired pages that the crawler acquires successfully, while the second one is the degree to which the acquired page snapshots remain up-to-date, relative to the current "live" web copies. Such key features are important both for the presented architecture and for web archiving. This latter is the process of collecting portions of the web to avoid the loss of information due to pages' modifications, content additions and pages' removals. In this way the information can be preserved for future researchers, historians and the public. The biggest web archiving organization performing a large-scale effort in web archiving is the Internet Archive [10]. Its estimated digitalized collection's size is in the order of 10 Petabytes. The Internet Archive has designed its own web crawler, called Heritrix [9], which has been able to crawl about 80 Terabytes of WARC (Web ARChive file format) during a period of nine months between March and December 2011 [17]. The crawling results have produced captures of about 2.7 billion URIs, whose 2.2 billion unique, including also images, flash and videos. The crawled data has been made available for viewing through the Waybach Machine, named by the Internet Archive as three-dimensional index. However, the Internet Archive is not the only organization involved in the campaign of maintaining universal access to all knowledge: national libraries, national archives and consortia are involved in archiving culturally important Web content. Unlike Internet Archive, other entities could be interested in time-awareness, namely taking in consideration the collections' time to improve response time and query throughput. Some of these, to realize time-aware web archives, employ Nutch and Solr technologies [5]. Our propose pushes on the use of these two technologies because they easily and deeply interact with each other. Moreover, Nutch is able to be run on top of Hadoop, allowing Nutch to exploit the promising features of MapReduce large-scale processing. Conversely, Heritrix was not born to cope with Hadoop, but rather it uses its own infrastructure. Clusters of Heritrix instances running across multiple machine are managed by a set of packages, called Heritrix Cluster Controller (HCC). The HCC is constituted by two main components: the controller itself and a client API for accessing it. Through the client API, the HCC offers a set of methods which perform the general functions of finding, listing, and invoking operations on single remote instances or groups of them. Nevertheless, a software integration is available to store crawled content in the Hadoop distributed file system, but not to run MapReduce jobs upon it. This is really the main difference between Apache Nutch and Internet Archive Heritrix: while Heritrix run multiple instance of itself on different servers, the same Nutch instance spreads the crawling job across multiple machines.

With respect to existing architectures described in [5], the one designed in this paper takes advantage of a data mining layer able to find out the possible correlations between indexed documents. Instead of having a searchable static document

collection, it provides machine learning tools to establish semantic relationships. This feature, in particular, can be useful both for analytics and for enhancing the user experience: in fact, search results can return documents matching the query as well as documents semantically related to the first ones. Similar architectures have been proposed both in [6] and [25] which present, respectively, a work-in-progress framework for lightweight semantic search with spatial support over data extracted from the web and an information retrieval architecture for aggregating, managing data and profile qualifications for IT solutions.

VII. Concluding Remarks

Understanding the influence relationships among updates published by a group of social communities is of primary importance in intelligence operations based on open sources. In this paper we sketched a model of interactions based on semantic causal relationships among updates and presented a Hadoop-based software architecture that is able to progressively scan the group of social communities, to store the huge amount of information obtained through such scanning and to mine the stored information to detect semantic causal relationship.

As a future work we plan to *(i)* define a formal model of social communities, *(ii)* study a quality metric which is able to evaluate the accuracy of the software system with respect to the semantic causal relationships detected by a software platform and *(iii)* analyze and visualize the linearizations of the partial order shown in Figure 1(b) in order to correlate the information contained in the updates belonging to the linearizations. The latter point could be considered as building a "video recorder" that is able to move back and forward to analyze each single linearization.

Acknowledgment

This work has been partially supported through the Italian MIUR PRIN project TENACE - National Critical Infrastructure Protection from Cyber Threats, and Italian MIUR Smart Cities and Communities project RoMA - Resilience enhancement Of Metropolitan Areas.

References

[1] V. Agneeswaran, "Big-data theoretical, engineering and analytics perspective," in *Big Data Analytics*, ser. Lecture Notes in Computer Science, S. Srinivasa and V. Bhatnagar, Eds. Springer Berlin Heidelberg, 2012, vol. 7678, pp. 8–15. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35542-4_2

[2] K. P. Birman, A. Schiper, and P. Stephenson, "Lightweigt causal and atomic group multicast," *ACM Trans. Comput. Syst.*, vol. 9, no. 3, pp. 272–314, 1991.

[3] A. A. Cárdenas, P. K. Manadhata, and S. Rajan, "Big data analytics for security intelligence," Big Data Working Group, Cloud Security Alliance, Tech. Rep., September 2013.

[4] E. Chickowski. (2013) A case study in security Big Data analysis, Dark Reading. [Online]. Available: http://www-01.ibm.com/software/data/bigdata/

[5] M. Costa, D. Gomes, F. Couto, and M. Silva, "A survey of web archive search architectures," in *WWW '13 Companion*, 2013, pp. 1045–1050.

[6] S. Dlugolinsky, M. Seleng, M. Laclavik, and L. Hluchy, "Distributed web-scale infrastructure for crawling, indexing and search with semantic support," *Computer Science*, vol. 13, no. 4, 2012. [Online]. Available: http://journals.agh.edu.pl/csci/article/view/42

[7] Gartner Inc. (2013) IT Glossary - Big Data. [Online]. Available: http://gartner.com/it-glossary/big-data/

[8] IBM. (2013) IBM Big Data Platform. [Online]. Available: http://www-01.ibm.com/software/data/bigdata/

[9] Internet Archive. (2011) Heritrix. [Online]. Available: http://crawler.archive.org/index.html

[10] ——. (2014). [Online]. Available: https://archive.org/index.php

[11] P. S. Kumar, M. Deepak, and R. B. Keshari, "Integration of web mining and web crawler: Relevance and state of art," *International Journal on Computer Science and Engineering*, vol. 2, no. 3, pp. 772–776, 2010.

[12] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[13] MapR Technologies. (2014) MapR. [Online]. Available: http://www.mapr.com

[14] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: Interactive analysis of web-scale datasets," in *Proc. of the 36th Int'l Conf on Very Large Data Bases*, 2010, pp. 330–339. [Online]. Available: http://www.vldb2010.org/accept.htm

[15] C. Olston and M. Najork, "Web crawling," *Found. Trends Inf. Retr.*, vol. 4, no. 3, pp. 175–246, Mar. 2010. [Online]. Available: http://dx.doi.org/10.1561/1500000017

[16] L. Rodrigues, R. Baldoni, E. Anceaume, and M. Raynal, "Deadline-constrained causal order," in *ISORC*, 2000.

[17] A. Rossi. (2012) 80 terabytes of archived web crawl data available for research, internet archive blogs. [Online]. Available: http://blog.archive.org/2012/10/26/80-terabytes-of-archived-web-crawl-data-available-for-research/

[18] The Apache Software Foundation. Apache Storm. [Online]. Available: http://storm.incubator.apache.org/

[19] ——. (2011-2012) Apache Solr. [Online]. Available: https://lucene.apache.org/solr/

[20] ——. (2013) Apache Nutch. [Online]. Available: https://nutch.apache.org/

[21] ——. (2013) Apache SolrCloud. [Online]. Available: https://cwiki.apache.org/confluence/display/solr/SolrCloud

[22] ——. (2013) NRT, Replication, and Disaster Recovery with SolrCloud. [Online]. Available: https://cwiki.apache.org/confluence/display/solr/NRT,+Replication,+and+Disaster+Recovery+with+SolrCloud/#DisasterRecoveryforanNRTsystem

[23] ——. (2014) Apache Mahout. [Online]. Available: https://mahout.apache.org/

[24] ——. (2014) Shards and Indexing Data in SolrCloud. [Online]. Available: https://cwiki.apache.org/confluence/display/solr/Shards+and+Indexing+Data+in+SolrCloud

[25] J. Woo, "Information retrieval architecture for heterogeneous big data on situation awareness," *Advanced Science and Technology*, vol. 59, pp. 113–122, 2013.