TransInfo
Tier I University Transportation Center

FINAL REPORT

An Evaluation of Knowledge Discovery Techniques for Big Transportation Data

September, 2018

Kevin Majka, Ph.D., Senior Research Scientist, CUBRC

Prepared by:
CUBRC
4455 Genesee St.
Buffalo, NY 14225

Prepared for:
Transportation Informatics Tier I University Transportation Center
204 Ketter Hall
University at Buffalo
Buffalo, NY 14260

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle An Evaluation of Knowledge Discovery Techniques for Big Transportation Data		**5. Report Date** September 2018
		6. Performing Organization Code
7. Author(s) Kevin Majka		**8. Performing Organization Report No.**
9. Performing Organization Name and Address CUBRC 4455 Genesee St. Buffalo, NY 14225		**10. Work Unit No. (TRAIS)**
		11. Contract or Grant No. DTRT13-G-UTC48
12. Sponsoring Agency Name and Address US Department of Transportation Office of the UTC Program, RDT-30 1200 New Jersey Ave., SE Washington, DC 20590		**13. Type of Report and Period Covered** Final Report June 2016 - September 2018
		14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract
In an ever growing internet of things (IoT) environment the amount of data produced and available for analysis is growing exponentially. In order to access and process these data sources new methods and tools are needed that can sufficiently process big data as well as ensure the quality and completeness of the data. For researchers and practitioners the use of technology should enhance their workflows and enable questions to be answered faster using all available relevant data. An intelligent architecture or software stack enables the data self service in a transparent way. This research proposes a prototype system that would enable researchers to efficiently work with 'Big Data' sources.

17. Key Words	18. Distribution Statement
Big Data, Informatics, Transportation Research	No restrictions. This document is available from the National Technical Information Service, Springfield, VA 22161

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	26	

Table of Contents

1 Introduction

In an ever growing internet of things (IoT) environment the amount of data produced and available for analysis is growing exponentially. These sources of data include those traditionally associated with transportation safety, such as crash, volume, and speed data but are increasingly becoming available through alternative sources like in vehicle telematics systems, cell phones, and vehicle to vehicle communications. In order to access and process these data sources new methods and tools are needed that can sufficiently process big data as well as ensure the quality and completeness of the data. For researchers and practitioners the use of technology should enhance their workflows and enable questions to be answered faster using all available relevant data. From a data science perspective a typical work flow involves: forming a hypothesis; collecting data; cleaning, aligning and transforming the data; query against the data; running analytics or models against the data; and visualizing or presenting the results. In addition, researchers should be able to extract data utilizing tools they are comfortable with. An intelligent architecture or software stack enables the data self service in a transparent way.

This research proposes a prototype system that would enable researchers to efficiently work with 'Big Data' sources. The goals of the system are: the use high-performance data processing technologies; batch, near real-time, and real time data processing; minimization of data movement to and from the client; rapid visualization of information on maps, tables, and graphs; the use of standardized web services to access information; and finally the use of open-source technologies that are free for production use. Two main problems that confront transportation researchers utilizing 'Big Data' are: (1) data alignment and (2) advanced analytics. Data alignment is the process of quickly aligning the terminology, semantics, and architecture of different data sets for the purpose of creating a unified common model for queries and analysis. A common data model motivates the development of advanced analytics algorithms that can effectively draw inferences across multiple aligned data sets. These algorithms go beyond extracting strings from text and instead extract meaning from information.

1.1 Infrastructure and Data Store

The base infrastructure and data store provide the foundation for overall prototype system. For development purposes, open source operating systems provide greater distribution flexibility. Linux based operating systems are robust and have large, active user communities. As a data platform, Hadoop[1] has become the de-facto standard for handling massive amounts of structured data. The main advantages of Hadoop include: plentiful data processing tools; durable, fault-tolerant, distributed data storage; scalable storage and processing; advanced storage and compression algorithms; an extremely active user community with new tools, projects, and research coming out every day. This prototype system proposes the use of the CentOS[2] operating system, Hortonworks HDP[3] data platform, and Apache Ambari[4] for the data platform management. These tools are described in Table 1.

Table 1. Infrastructure and Data Store Tools

Tool Category	Tool Name	Description
Operating System	CentOS	"CentOS is a Linux distribution that attempts to provide a free, enterprise-class, community-supported computing platform which aims to be functionally compatible with Red Hat Enterprise Linux"
Data Platform	Hortonworks HDP	A managed open-source Hadoop Distribution. A distribution of Hadoop will contain the tools and software of the Hadoop Ecosystem guaranteed to work well with each other.
Data Platform Management	Apache Ambari	An integrated tool "aimed at making Hadoop management simpler by developing software for provisioning, managing, and monitoring Apache Hadoop clusters."

1.2 Data Collection

When we obtain data it usually comes in many different formats such as structured text reports (Tweets, News Reports), semi-structured reports in tabular formats (Police Reports, Sensor Data), or fully structured relational or geospatial databases (Crash Databases, Pavement Conditions). This data can be static or dynamic in nature and can be obtained in many different ways. The data can be accessed from historical archives (Crash Databases, AADT), batch processed (Incident Reports, Weather Reports), or ingested form streaming real or near-real time data sources (Tweets, Speed Sensors). Various tools that allow this data to be extracted, transformed, and loaded in to the data store are necessary. Three such tools that allow for the efficiently handling of data are Talend[5], Pentaho Data Integration[6], and Informatica[7]. Each of these tools is described in Table 2.

Table 2. Data Extract, Transform, and Load Tools

Tool Category	Tool Name	Description
Data Ingestion	Talend	Talend is a multi-purpose Extract, Transform, Load (ETL) tool that can read data from a variety of resources (file, database, web services) and load it into a target locations/formats
	Pentaho Data Integration	"Pentaho Data Integration consists of a core data integration (ETL) engine, and GUI applications that allow the user to define data integration jobs and transformations"
	Informatica	"Informatica is focuses on data integration, ETL, cloud integration, complex event processing, masking, data quality, data replication, and virtualization to maintaining enterprise-wide data warehouses"

1.3 Data Sources

Our proposed technical approach recognizes the TRIP system must be capable of servicing a large number of disparate transportation safety-related data sets. Examples of these data sets are illustrated in Figure 1. Identified in the figure are traditional archival crash data (e.g., HSIS, NASS, FARS, LTCCS, NMVCCS), in depth crash and injury data (e.g., CIREN), and naturalistic driving data (e.g., SHRP 2 NDS & 100 Car Study). In addition, the figure identifies weather data, roadway characteristic data, and traffic data that are currently available but not easily linked to the crash data bases. Finally, the figure recognizes the potential great utility of Advanced Automated Crash Notification telemetry data provided in real time by crash vehicles (i.e., OnStar and BMW Assist) to private call centers. This data is currently available through the Condition Acquisition Reporting System (CARS) in selected states and provides very accurate crash times as well as vehicle based measurements of crash characteristics (i.e., Principal Direction of Force and crash delta velocity). The AACN data is expected to be more widely available when the Next Generation 911 system is implemented.

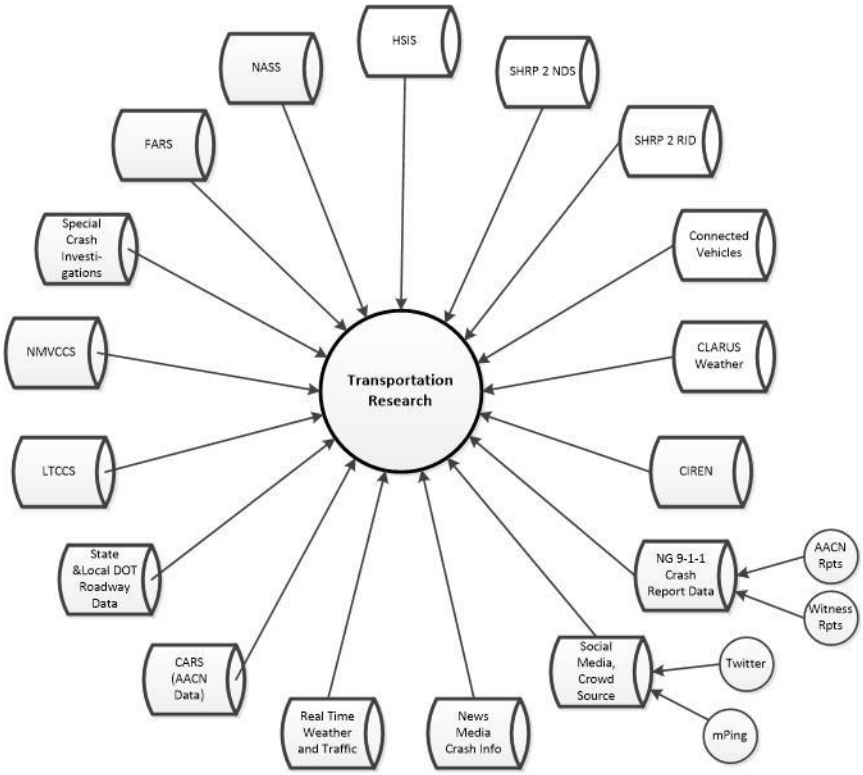

Figure 1. Examples of Sources to Support 'Big Data' Transportation Safety Analyses

In the following sections, the major components (Process, Query, and Analytics) of a core workflow and supporting systems will be explained in detail and specific recommendations will be given for the use of open source software. An Overview of a typical workflow with associated tools is illustrated in Figure 2.

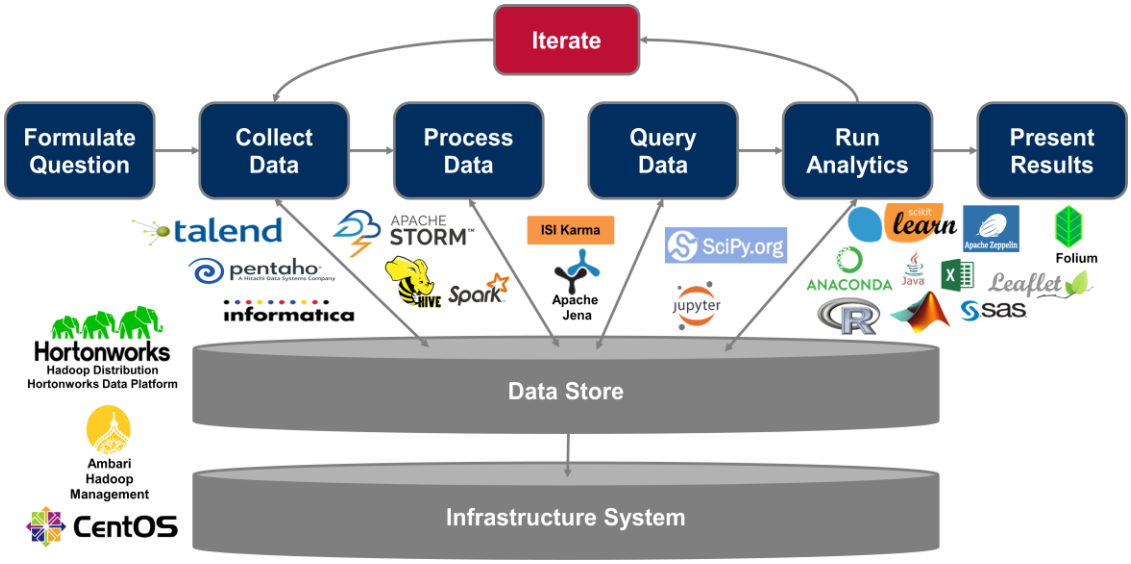

Figure 2. Typical System Architecture

2 Data Processing

This prototype system assembles a host of state-of-the-art systems developed to realize the goal of data alignment for large-scale, disparate data sources. At its core, it uses a suite of layered ontologies to serve as targets for information extraction processes and, importantly, establish a semantic layer where singular queries may be issued across multiple sources. The ontologies provide a rich set of concepts that represent information of interest to analysts

2.1 Ontologies

Ontologies are formal descriptions of domains of interest. Two common uses of ontologies are as a vocabulary that can express the content of discrete data sources, or as a knowledge base that can be queried for facts about the domain of interest. Note that these two uses are not exclusive. Ontologies can be expressed in a variety of ways, ranging from natural language to any of the knowledge representation languages. Within the system, the ontologies are represented in the Web Ontology Language (OWL). The World Wide Web Consortium (W3C) is encouraging the semantic web community to standardize ontology development using OWL, and this effort leads to greater interoperability both in terms of the ontologies themselves, but also in terms of the applications (editors, reasoners, stores, and query engines) that operate upon them.

In order to provide a vocabulary that is used to re-express the content of a data source, ontologies contain classes, which are representations of the types of objects and events that comprise the domain of interest, and properties, which are representations of the relationships that exist between the objects and events of the domain. Classes and properties are organized into two taxonomies: hierarchies based on subclass and sub-property relationships. These taxonomies, if they are developed using known best practices, can improve the results of queries by organizing all information about a domain type and its subtypes (e.g., vehicle: ground vehicle, truck, armored truck) into a single branch of the taxonomy.

To serve as a knowledge base, ontologies must contain assertions that describe the known facts of a given domain. In OWL, such assertions take the form of what are called restrictions, which are statements of either necessary or necessary and sufficient conditions of class membership. A few examples of these may be helpful: If person is a type represented in the ontology by a class, then an assertion describing a necessary condition associated with that class might be “Every person has a weight.” If populated place is a type represented in the ontology by a class, then an assertion describing necessary and sufficient conditions associated with that class might be “Every populated place is a geospatial region that is the location of some population.”
OWL is sufficiently expressive for complex assertions formed through the use of combinations of set-theoretic operators (intersection, union, and complement) and numeric ranges. The information stored in these restrictions can be of value both for human and machine consumers.

The implementation of such restrictions can take other forms, including SPARQL Protocol and RDF Query Language (SPARQL) construct queries, which can have advantages over their implementation in OWL. One example of such an advantage is that a construct query can be designed to create an instance of an entity based upon the information expressed in the restriction. For example, a construct query could create an instance of a population located at the geospatial region that is a populated place. This instance could then be linked to other information sources that describe the demographics of that population.

2.2 Data Alignment

There are two modules used for the alignment of structured and unstructured data into the ontology. Key to the process of structured data alignment is the introduction of tools that expedite alignment and make it possible to align hundreds of data sources to a singular ontology model. To this end, structured data alignment is quickly becoming a semi-automated process. Alignment of unstructured data uses a unique natural language processing approach that extracts rich entity information. The approach focuses on relationship-chaining, where sets of known relationship patterns combine to elicit valuable semantics from data.

2.2.1 Structured Data Alignment

Structured data sources are relational tables, meaning that each value in a row has a known relation to some other value in that row. For instance, if there is a PersonName column and a Gender column, it may be the case that the gender value is attributed to the Person; if there is a DateOfBirth column, it may be the case that the date of birth value is associated to the Person; and so on. Given the known dependencies that exist in structured data, the concepts and their relations to each other can easily be aligned to the ontology. This alignment relies on a D2RQ[8] mapping language document. In a D2RQ map, there are two kinds of sub-maps: Class Maps, which associate column headers or categorical concepts to ontological types, and Property Bridges, which create the relations that exist between Class Maps. Each statement of a sub-map is a triple, which includes the map of interest, an attribute category, and the value of the attribute.

2.2.2 Unstructured Data Alignment

Unstructured data sources can contain documents, news articles, interpersonal correspondence, field reports, slideshows, etc. Unlike structured data, where relations and concepts can be identified and extracted via a correlative map resource, unstructured data is sent through a natural language processing (NLP) pipeline. In an NLP pipeline, discrete, sequential analytical engines iterate over the data with the goal of "parsing" the data. The output of the first analytical engine serves as input to the second analytical engine, and so on. Sentences, phrases, and words are extracted, indexed, assigned parts of speech and syntactic categories, and then fit to semantic templates based on their environment. The output of the NLP pipeline is aligned to the ontology in one of two ways: via token-based mappings or context-based mappings.

A token-based mapping is a 1:1 or n:1 mapping, where a particular word has one, unambiguous mapping into the ontology. Note that it is not necessary for the ontological class to have one, unambiguous token associated with it, hence the n:1 mapping. Semantic distance (a similarity measure) is also included in token-based mappings, which means that conceptual subtypes and supertypes are considered when a mapping is being identified. For example if "Corvette" does not have a direct mapping into the ontology, the mapper employs the WordNet hypernym/hyponym concept hierarchies to find a suitable mapping. In this case, maybe the token "Corvette" will be mapped to the ontology class automobile:Chevrolet, as Chevrolet is a supertype of Corvette, and, crucially, the semantic distance between the two concepts has been calculated to be within some preset similarity threshold, to avoid arbitrary concept replacements. Token-based mappings are executed via look-up lists.

In the case of ambiguous tokens (tokens with more than one meaning or possible ontological mapping), the type of the entity is considered, as is the surrounding context. For example, the numerical entity "29" will by default be mapped to an ontology class such as Amount. However, if the number entity occurs inside the semantic relation IS_AGE_OF, it will then be mapped to an ontology class such as Chronological_Age, as the surrounding context suggests that the number represents some entity's age rather than a volume or amount. Context-based mappings are executed via conditional statements.

2.3 Ingestion Processes

Two high-level components responsible for ingesting data are proposed. The component dedicated to Unstructured Data Alignment is provided by the Lymba Corporation[9], while the component for Structured Data Alignment is provided by ISS, Inc. There is boilerplate code for invoking each of these components in the same way, which is replicated throughout the code base. A higher-level component, known as the Ingestion Service, has been developed to provide a unified interface to processing both Structured and Unstructured Data Alignment. The Ingestion Service has been defined with input from various team members and is flexible enough to provide all features that are required.

The service has been designed to be application-agnostic, and can be run in a traditional standalone Java application. Additionally, platforms such as Hadoop, Storm, and Web Application servers are fully compatible with the service, and take advantage of the modular nature of the Ingestion Service to produce specialized implementations of individual components. The threading model of the Ingestion Service expects that only one document is transmitted at a time. However, the service has been designed so that users can create as many instances of the service as needed to attain the parallelism required. These parameters are often set by the platform in which the service is contained; therefore, not imposing them in the framework allows the greatest flexibility.

The Ingestion Service takes in an ingestion request in the form of a uniform resource identifier (URI), which uniquely identifies a document from the data sets. It is then passed to the primary alignment components, in a process known as primary ingestion. Depending on whether the document is structured or unstructured, it then moves through the appropriate data alignment. Then a series of post-processors are launched, which are responsible for performing advanced analytics on the data that has been aligned to the data model. The final result is stored in the storage location specified by the Ingestion Storage component. An overview of this process is illustrated in Figure 3.

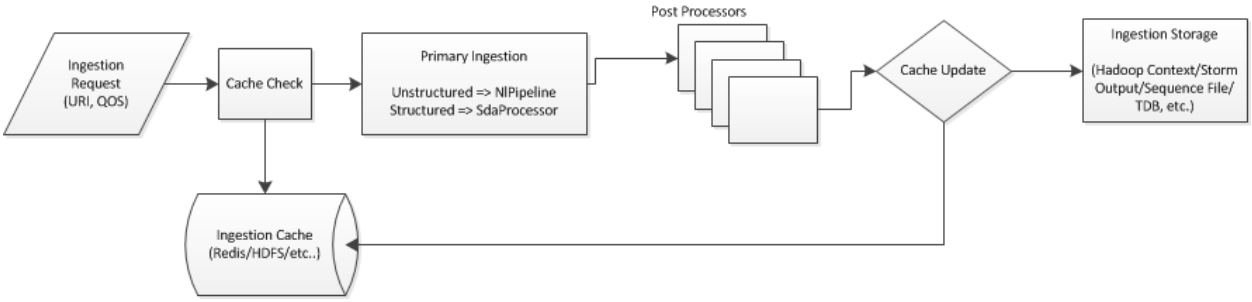

Figure 3. Ingestion Processes

The Ingestion Service Framework is a modular service that is constructed by the user through simple Java object operations, or assembly through Spring Context. Several factory objects have been developed to aid in the construction of common Ingestion Service configurations. The components that comprise the Ingestion Service are defined by interfaces, which make it easy to create new implementations that extend the functionality of the Ingestion Service.

2.4 Ingestion Elements

Ingestion Metric Services include everything from log results to specialized implementations, such as the Hadoop Context Ingestion Metrics Service, which takes in the Hadoop Context and enables data to be displayed in the MapReduce Web Interfaces and Reports. Once data have been aligned to the alignment data model, post-processing analytics such as Location Resolution and Entity/Event Coreference Resolution can be launched against each document before storing the data in the target ingestion storage repository. This series of post-processors is a chain, so the results of a previous post-processor are fed into the next post-processor. There are various data storage formats available. Each implementation of the Ingestion Metric Services handles the storage that would otherwise be duplicated across the code base.

2.4.1 Apache Spark, Hive, and Storm

Apache Spark[10] can be used as a general processing engine to create and schedule alignment and transformation jobs. Spark allows for in-memory processing, allowing for faster data operations. A Spark job is defined by 2 basic parameters; the number of executors and memory per executor. To allocate these executors the Hadoop Resource Negotiator called YARN (Yet Another

Resource Negotiator) can be utilized. This process allows for jobs to be easily scaled horizontally across available resources. Higher level data processing technologies such as Apache Hive[11] can be utilized to provide a more user friendly SQL type query language.

In addition, Apache Storm[12] can be utilized to handle streaming real-time data processing. Apache Storm is a distributed, high-throughput, real-time processing framework developed by Twitter. A Storm cluster is capable of executing in excess of 1 million tuples per second per node. Storm is comprised of three main elements: Spouts, Bolts, and Streams. A Storm workflow is a called a Topology. Spouts are the source of data in the topology. Typical implementations of Spouts draw from data sources via Hadoop Distributed File System (HDFS), Kafka, or Redis. The Storm ingestion process consists of a series of steps. The input to the process is a list of queries to execute, or text/documents to process. The output is a single resource description framework (RDF) store, with resolved entities from all processed entities from all processed documents. Although there are a number of steps, nearly all steps can be executed in parallel using the Storm framework. The only two steps which cannot be processed in parallel are Entity Resolution and Document Deduplication, as both of these steps require access to all data in order to execute. Fortunately, these two steps comprise only a small portion of the entire execution time. A summary of these tools is provided in Table 3.

Table 3. Data Processing Tools

Tool Category	Tool Name	Description
Data Processing	Apache Spark	A general purpose, distributed data processing framework. Apache Spark is well-suited to iterative and machine learning applications because of it's in memory architecture and easy processing API
	Apache Hive	"Data warehousing software that facilitates querying and managing large datasets residing in distributed storage. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language"
	Apache Storm	"Apache Storm is a free and open source distributed real-time computation system to reliably process unbounded streaming data"

2.5 Hadoop

To more easily develop modular and reusable Hadoop jobs an application development framework called Cascading is utilized. Cascading allows a user to assemble a MapReduce job using generic functions called Pipes, and link them together without requiring any a priori understanding of how to write a MapReduce job or how to optimally assemble the Map and Reduce functions of the job. Cascading has many built-in functions, including joining and splitting pipes, filtering values, mathematical operations such as Sum, Average, Count, Min, and Max, and various Regular Expression functions. Cascading also abstracts the source and destination of data through Taps. Taps allow the data to be read in from or to any HDFS location.

It is easy to extend the Taps implementation to use a new source of synchronization such as a SQL database or web feed. In this system, the major logical components are grouped into cascading subassemblies, which represent the groupings of components that are executed together. Also utilized are the Full Ingestion and Data Ingestion subassemblies.

2.5.1 Full Ingestion Subassembly

The purpose of the full ingestion subassembly is to read in a data set from a data service, process the entire corpus of data, and cache the Resource Description Framework (RDF) documents in the RDF caching layer. The process works in two steps: a quick ingestion and a complete ingestion. Quick ingestion will attempt to process the documents as quickly as it can, using the default timeout of the natural language processing (NLP) pipeline. The complete ingestion will extend that timeout as necessary, in an attempt to re-process documents that may have failed in the previous step. The full ingestion subassembly will first divide the data set from the data service with one of two strategies: first, there is an automatic method where the number of slots used on the MapReduce cluster is specified, and then the data sets are distributed as evenly as possible across the processors. With this method, the data sets that are unrestricted and freely queried will be fully processed. The second method is to parameterize the data set names and the number of documents to ingest. With this method, both restricted and unrestricted data sets can be queried.

2.5.2 Data Ingestion Subassembly

The data ingestion subassembly is the main subassembly responsible for querying, retrieving, and creating an RDF triple store resulting from a user-supplied query. The source Tap is a specially-designed Tap that processes a data set from the data service. The sync Tap will store the documents in the Apache Tuple Database (TDB) store. The main subassembly is similar to the quick ingestion subassembly, which attempts to process documents as quickly as possible. Additionally, this subassembly can be utilized instead of the Storm ingestion workflow, if the target environment does not have Storm installed.

3 Query Data

In order to provide analysts with and efficient way to query the data a multifaceted user interface (UI) is proposed. The UI encourages analysts to build, test, and refine high-precision queries before a workflow is finally executed against the system. To improve accuracy and insight the UI is also built utilizing a Resource Description Framework (RDF).

3.1 User Interface

The proposed UI is meant to be a sophisticated, intuitive search tool designed to deliver raw data to analysts as quickly as possible. The UI design employs an easy-to-use, simplified window into the data, and is built on the Lucene Syntax query language. The UI is unique because it assists the user in composing quality, complex queries, thanks to its extensive and configurable topic and filter libraries. The UI provides snapshots of each query's coverage across the available data sets, and returns sample results in seconds.

3.1.1 Search Interface

The Search Interface includes a query builder and pre-defined topic filters. The query builder accepts Boolean syntax operators such as AND, OR and NOT. Queries can consist of single terms, or multi-word phrases inside double quotes. Additional Lucene query syntax features, such as proximity searching, are supported in the query builder. Filters, which are sets of pre-defined query expansions, aim to assist the user with building more robust queries, and are available for a number of topics. Applying a filter to a query saves analysts time at the query design stage, and furthermore, analysts are able to create, modify, and delete custom filters.

With respect to query coverage, the analyst has the option to view the number of documents in each data set that match a query, prior to actually executing the query on the data. This function is especially useful for identifying which data sources are the most relevant for a particular query, and for tailoring queries to avoid being too broad or narrow with a search. All queries that are issued through the Search UI, whether executing a coverage count or a full-scale search, are subject to pre-processing and query syntax validation. Pre-processing allows for the inclusion of custom syntax, which is converted to the native Lucene, while validation ensures that queries adhere to proper syntactic form.

3.1.2 Results Interface

The Search Results Interface is composed primarily of the document results pane. Inside the document results pane, each document result is enumerated, and includes a reference to the data source name, as well as a headline or filename for that result. All terms that match the original query are highlighted in red. The user can choose to download the original document, if so desired. Note that results from unstructured data sources will display differently than results from structured data sources. Unstructured results will appear as a series of snippets, or zones around

the sections of text that match the query. Structured results will appear as a table, and only the cells that match the query will be displayed from a given row.

3.2 Resource Description Framework Translation

This proposed system is designed to handle large quantities of heterogeneous data. To be useful, the knowledge present in such data must be processed, which requires a robust framework for storage, organization, and retrieval. The RDF format is a straightforward approach to entity-relationship modeling, and is therefore well-suited to the goals of knowledge representation and the semantic web. The use of RDF, which, together with the linked data paradigm, is an increasingly popular framework of choice for such systems. RDF is a directed graph representation of resources, capable of managing information about nodes and arcs within the graph.

3.2.1 Resource Description Framework Storage

Storage and querying of arbitrarily large graph structures, as depicted below, are critical pieces of any system that performs alignment across multiple data sets. Identification of a suitable data store for RDF graphs that is both open-source (i.e., no commercial licensing required) and with fast read/write times was an open challenge at the outset of the project in 2010. Many data store solutions were implemented and evaluated in the first two phases of this project, which ranged from relational database stores to HBase table schemas. None were suitable solutions and forced a re-examination of current open-source projects to develop a blend of data stores suitable for all use-cases.

The current implementation is a tiered graph storage solution. The tiers can be conceptualized as small, medium, and large solutions to accommodate many different graph sizes. Small graphs are those that can fit into a system-defined RAM size. For example, the system can be configured to store graphs smaller than a particular threshold (e.g., 1GB, 2GB, 4GB, etc.) on HDFS. These graphs are loaded directly to RAM when queries are read or executed. Medium graphs are those that can fit into a TDB store. TDB is an Apache open-source solution for storage and query access of RDF. The upper bound for TDB graphs is roughly the hard-disk size of any given node in the cluster. Large graphs are those that can grow arbitrarily large and ultimately exceed the upper bounds of small and medium graphs. A potential solution to this problem is the utilization of Hive.

The small and medium graph solutions are both fully implementable. Both boast high SPARQL querying speeds and do not require a server other than HDFS to store the data. The Hive solution will implement the API and directed the use of Hadoop rather than a relational database. Hive is a particularly attractive solution because as the amount of data to be stored increases, the system will only need to increase its storage hardware to maintain the same efficiencies for reading and writing queries. All RDF storage solutions should be designed to support the full SPARQL 1.1

specification. Any RDF graph structure stored in the system has a complimentary SPARQL endpoint where the graph can be queried using the latest SPARQL specification. The Hive large graph storage solution is meant to uphold this contract for arbitrarily large-sized graphs.

3.2.2 Apache Tuple Database

The Apache Tuple Database (TDB) is a fast, indexed, filesystem-based triple store created by the Apache Jena[13] project. For both ingest processes (Storm and Hadoop), the RDF that is generated is written to the TDB store, then compressed and copied to the HDFS. Later, other processes retrieve the triple store from the HDFS, uncompress it, and query against it. If the triple store is modified, the updated triple store can be re-compressed and stored on HDFS. The maximum scalable size of a TDB store is the size of the hard drive that is contained on a single node.

3.2.3 Apache Hive

The TDB RDF querying and storage solution provides quick results when the data does not exceed a few gigabytes. However, it operates only on a single machine (sequentially) and requires preemptive indexing, and thus it is not scalable. The need for a more scalable solution exists for any system with a large-enough volume of data. The Jena SDB and Hive-based RDF storage solution is designed to fill this void, and will prove useful when there is too much RDF and it is no longer feasible for one machine to manage it. Although open-source GPL solutions exist, none of them integrate natively with Hadoop, and thus require their own environment to run. The Hive solution is designed to work on preexisting Hadoop clusters.

SPARQL is supported by Jena SDB. Jena SDB is an open-source Apache project which provides a translation layer from SPARQL to SQL (the traditional database querying language). Since Hive operates on a SQL-like query language, called HiveQL, extending Jena SDB to work with Hive is one approach to handling vast amounts of RDF in parallel. Using Hive as the back-end storage allows us to leverage its parallel processing querying engine to complete SPARQL queries. This store is referred to as the Hive Triple Store, though in reality, it is Cloudera's Impala query engine doing the heavy-lifting. Impala has been proven to be a plug-and-pay switch-over from the Hive query engine and has resulted in big performance increases. The Hive query engine is dependent on MapReduce, so it is unreasonable to expect it to work in real time. Impala still operates in parallel, uses the same query language, and reads the same data as Hive, but it was designed for real-time analytics, as opposed to batch-oriented queries. To clarify some terminology, Impala could be viewed as just the query engine, whereas Hive could be viewed as the entire data warehouse.

This solution would be optimal when the volume of RDF is such that TDB does not have enough space to accommodate the graph, or works too slowly on it. The Hive Triple Storage solution can be used to parallelize the query processing, RDF storage, and RDF access. The integration of this solution into system is fairly straightforward and can be done in a dynamic way, such that when

the large RDF graph is detected, it is ingested into Hive instead of TDB. This component exists as an alternative storage and query method to TDB, while still maintaining the same triple store abstraction. In addition to making Jena SDB compatible with Hive, steps are being taken to improve upon this component to address newfound complexities with storing and querying such large amounts of data. The improvements show great performance gains over the baseline system. Additionally, a more capable SPARQL-to-SQL translation layer is in the works, which pushes more of the processing to the data, as opposed to pulling the data to the processing. Doing more work on a data-local server allows for dramatically reduced network traffic.

Most systems store the ontologies with the data, but this proposed system separates the ontologies from the data. This change allows the system to pre-process the SPARQL query to explicitly represent any desired ontological patterns. If the ontologies were stored with the data, too much time would be spent weeding through the volume of data to access a simple ontology concept. Pre-processing allows the system to look up ontology concepts ahead of time to avoid getting lost in the data. The data can be compressed in both the raw text format and in sequence files. These options have proven to yield less CPU usage, less storage space usage, and better query time scaling.

3.3 SPARQL Protocol And RDF Query Language

Because RDF data is interpretable as a graph, SPARQL Protocol And RDF Query Language (SPARQL), whose queries are expressed as descriptions of a graph, is the standard RDF query language according to the W3C. A SPARQL query takes the form of a subject-predicate-object triple, much like the triples found in RDF, except that, where the elements of an RDF triple are assertions, the elements of a SPARQL query can be either literals or variables. Therefore, SPARQL queries might match zero or more subgraphs of the larger RDF graph.

3.4 TruST

TruST is integrated within the system by replacing the default graph matcher used by Jena's ARQ engine. ARQ consists of three main components. The SPARQL Parser (ARQ calls this "Parsing and Algebra Generation"), which reads the SPARQL query and devises a plan for its execution. The Query Executor ("OpExecutor"), which implements the plan generated by the SPARQL Parser by calling the graph matching component, and evaluating any required algebraic expressions (OPTIONAL, UNION, etc.). The Graph Matching ("Stage Generator") is the component which TruST replaces. The default component included with Jena executes a full depth first search on the RDF Graph to find matches to the query graph. TruST overrides this behavior to instead find approximate matches.

When a SPARQL query comes in to the system, the SPARQL Parser translates it into a series of steps for the query executor. Any such step that requires finding a graph match is delegated to the graph matching engine currently registered with ARQ. By default, ARQ uses a brute force

search, but TruST can be substituted here to gain approximate matching capabilities. The advantage of inserting TruST below the QueryExecutor is that any tasks related to features of SPARQL that are not graph matching, such as OPTIONAL, UNION, FILTER, etc., are still handled by the QueryExecutor. An overview of the data query process is presented in figure 4.

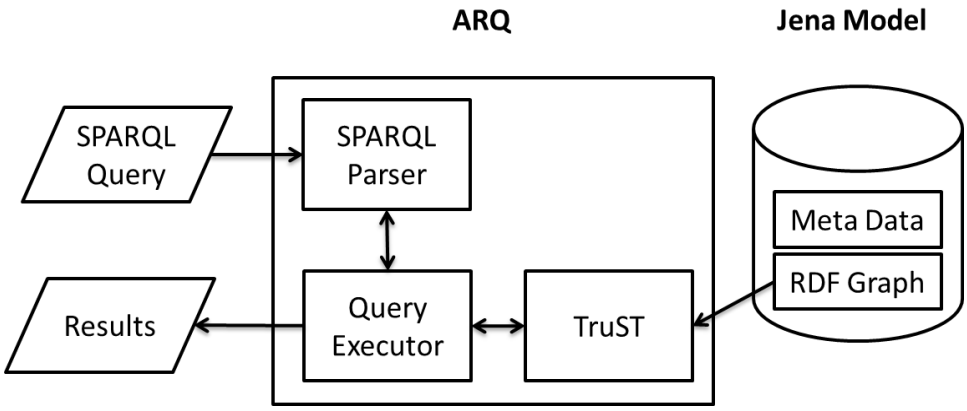

Figure 4. Data Query Process Overview

4 Analytics

Advanced analytics generate insights into data that enhance the richness of the output by exploiting targeted aspects of the data. This system was designed with these capabilities in mind. Coreference resolution analytics gather multiple references to a particular entity or event and associate them together based on a set of features and attributes that have been defined a priori. This system includes core coreference resolution analytics for entity and event resolution that feature the ability to customize attribute sets that are required for reliable entity coreference and disambiguation. Unlike many other approaches to coreference resolution, this approach applies probabilities of shared feature sets to entities to assert whether one entity is the same as another. This functionality is reliable within documents, across documents, and even across data sources. An added bonus to this approach is that social network algorithms can execute against all data sources, as entity resolution will prune out duplicate entities without over-populating a network graph.

4.1 Entity Resolution

Entity resolution allows an analyst to quickly perform a comprehensive search on a person and collect important attributes of that person without a manual search. The process begins with some defining information about a person of interest, such as that person's name, date of birth, and current or past locations. The entity resolution algorithm then searches the data sources based on this provided information and finds other mentions of the person, some of which will contain additional attributes not previously known to the analyst (e.g. education, marital status, etc.) These additional attributes will be collected and aggregated into a more complete representation of the person's true attributes. The resulting visualization may present this information as a table of person attributes, as a timeline of important events in that person's life, or as a map of locations that person has been.

The systems approach to entity resolution is modeled after the equality logic problem (closely related to the Boolean SAT problem). This model operates using only attributes decided to be strong identifiers, or a sets of attributes that collectively behave as strong identifiers (e.g. name, date of birth, and birthplace). Attributes that behave as strong dis-identifiers are used to detect inconsistencies with the strong identifier models. When a conflict in strong identifiers occurs (e.g. two people have the same name, date of birth, and birthplace but different genders), an attempt will be made to resolve this in a way most likely to reflect reality.

This research benefits the system and analyst by allowing new data sources to be quickly added to a resolution solution with minimal (or no) ground truth to train on, through the addition of new linking and disambiguation criteria. The capability to add such criteria on-the-fly also permits an analyst to adjust the solution in response to their findings; if an analyst discovers a strong way to link entities in the data, they can add this as a rule to the resolution system. Finally, by focusing

on strong linking models, it is easier for a user to visualize the resolution solution and follow pedigree information to determine why the system decided two entity mentions are coreferent.

The basic flow of the entity resolution system takes place in three steps. The first step reads RDF data and extracts entity attributes that will be used for resolution. The second step ("Entity Linker") uses these attributes to decide which entities might corefer. Finally, the Entity Disambiguation step takes the candidates from the Entity Linker, examines any conflicting information, then makes a final decision about coreference which it then writes back to the RDF store. An overview of the Entity resolution process is illustrated in Figure 5.

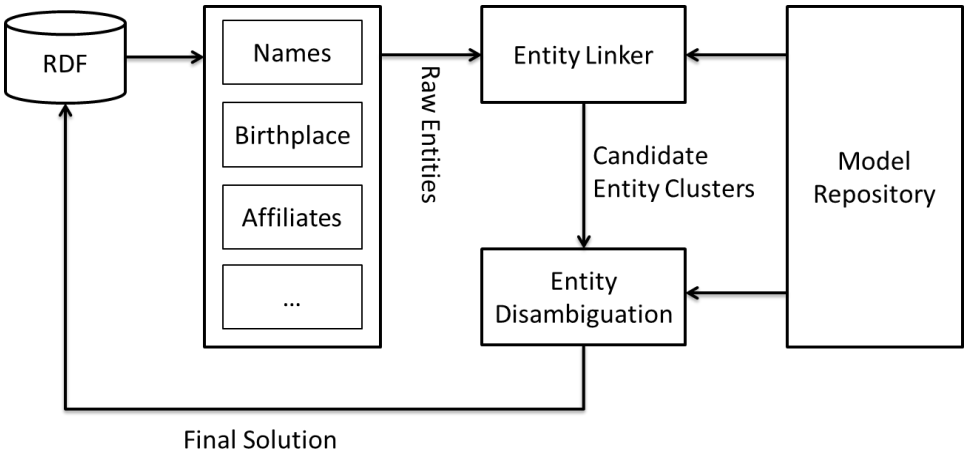

Figure 5. Entity Resolution Process

Three additional advanced analytics that are streamlined for the Cloud environment are also proposed. They are inexact (also called “dirty”) graph matching, location resolution, and social network analysis.

4.2 Inexact Graph Matching

SPARQL queries are expressed as a description of a graph, where some variables of interest are linked to other variables of interest by edges. Execution of a SPARQL query closely resembles the problem of graph matching, especially the problem of subgraph isomorphism, also known as exact graph matching. As with exact graph matching, SPARQL relies on clean data. Although SPARQL contains a number of features which make it flexible to some inconsistency, the data to be queries must be largely free of noise and errors. This requirement is in conflict with the reality that data processed from raw sources almost always contains some noise and errors. Regardless of its source, searching for results in noisy data is an important problem, as it improves the system’s ability to find relevant results in noisy data.

In exact graph matching, it is required that the query (“template” or SPARQL) graph matches a subgraph of the RDF (“data”) graph exactly; both the structure of the graphs and attributes of

nodes and edges must be exactly the same. Because of this, exact graph matching has limited opportunity to accommodate errors in the template of data graphs. With inexact graph matching, however, two graphs are no longer classified as the same or not the same, but rather placed on a continuous scale of how similar they are. Graph similarity is intended to capture how closely both the structure and attributes of two graphs match. In support of this, it is common to express the similarity between nodes (edges) from the template graph to nodes (edges) from the data graph as a numeric value and find a match using some objective function which favors selecting matches with high similarity. These similarity measures can be chosen following a probabilistic interpretation, however, this system currently focuses on manually selected similarity measures. The quality of a graph match is defined to be the sum of the similarities of all elements that are matched between the template graph and the data graph.

4.3 Location Resolution

Location resolution performs an invaluable task. Location designators are normalized by leveraging GeoNames in a manner such that they are resolved to the most specific named location and its area of relevancy, which is computed based on the population of a region in the world. The reason that such an approach is used is to ensure that locations indicated in various sources can be determined to have actually occurred at the same place. As an example, consider the crash of Colgan Air Flight 3407, which occurred in Clarence Center, which is a township near Buffalo NY USA. In early reporting, some sources did not know the exact location that the plane crashed--only that it was near Buffalo. By resolving these regions based on population, it can be determined that the area of relevancy for Buffalo actually overlaps that of Clarence Center, thus allowing for these locations to be considered similar in later analytics.

Location accuracy is paramount, and for this reason, named locations must be tied to a country, at the absolute minimum. An example as to why this requirement is enforced can clearly be illustrated when considering two locations with the same name (Paris, Ohio, United States; Paris, France). The problem described above does not exist when geo-coordinates are extracted from the data. When this type of location designator is used, even if the location does not resolve to populated region (country, city, or province), the surrounding region will be represented by a bounding box, which is approximately 500 square kilometers around the point of interest. The reason a box of this size is used is to account for geo-coordinates that actually point to an ocean or sea location.

With locations being resolved to English-language representations and the regions which encompass them, it is possible for the end-user to query the data either by name or by selecting a bounding box. This selection can be used to limit the data that the user must sift through. As in the Clarence Center example above, if one were to search the data for “plane crash” and restrict the location to the Buffalo, NY region, the matching results would only include data that related to the query in the specified region, and all plane crashes that did not have a location in the

Buffalo area would be filtered out. An overview of the location resolution process is illustrated in Figure 6.

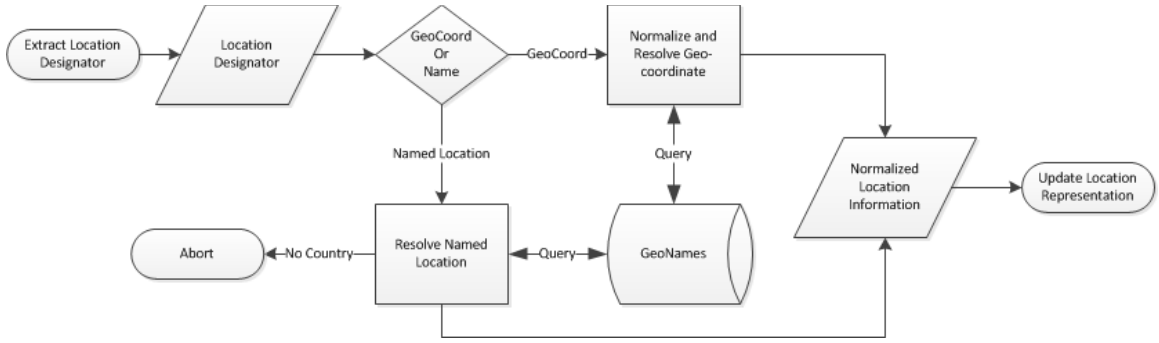

Figure 6. Location Resolution Process

4.4 Social Network Analysis

Social network extraction offers a means for a speedy investigation of individuals discovered during normal acquisition operations. For instance, an analyst prepares a report on a particular crash event and in doing so, uncovers previously unknown witnesses/acquaintances. This unknown individual could then be run through the social network extraction algorithm while the analyst spends his time on the more prominent persons. If the resulting network reveals some important connections, the analyst may wish to revisit the new associate and conduct further research.

Social network extraction across multiple, disparate data sources is a difficult endeavor. Existing extraction routines typically focus on singular, high-quality data sources where record-level metadata information enable relatively simplistic social network extraction (i.e., extraction of a network from a social media site where the numbers of posts and replies between accounts are indicators of active relationships). Less common are approaches that focus on the extraction of social networks from natural language text across multiple data sources, or the extraction of networks from classified data sources where message metadata coupled with network extraction from text may identify rich networks.

This proposed algorithm creates a social network for a person by repeatedly searching for new people that are discovered who have connections to the initial person of interest. The new persons who are discovered are then fed back into the algorithm in order to acquire more data, and the process is repeated on any person designators that are discovered in the network. Once no more new person designators can be discovered, or the requery limit is reached, the network is returned. An overview of the social network extraction process is illustrated in Figure 7.

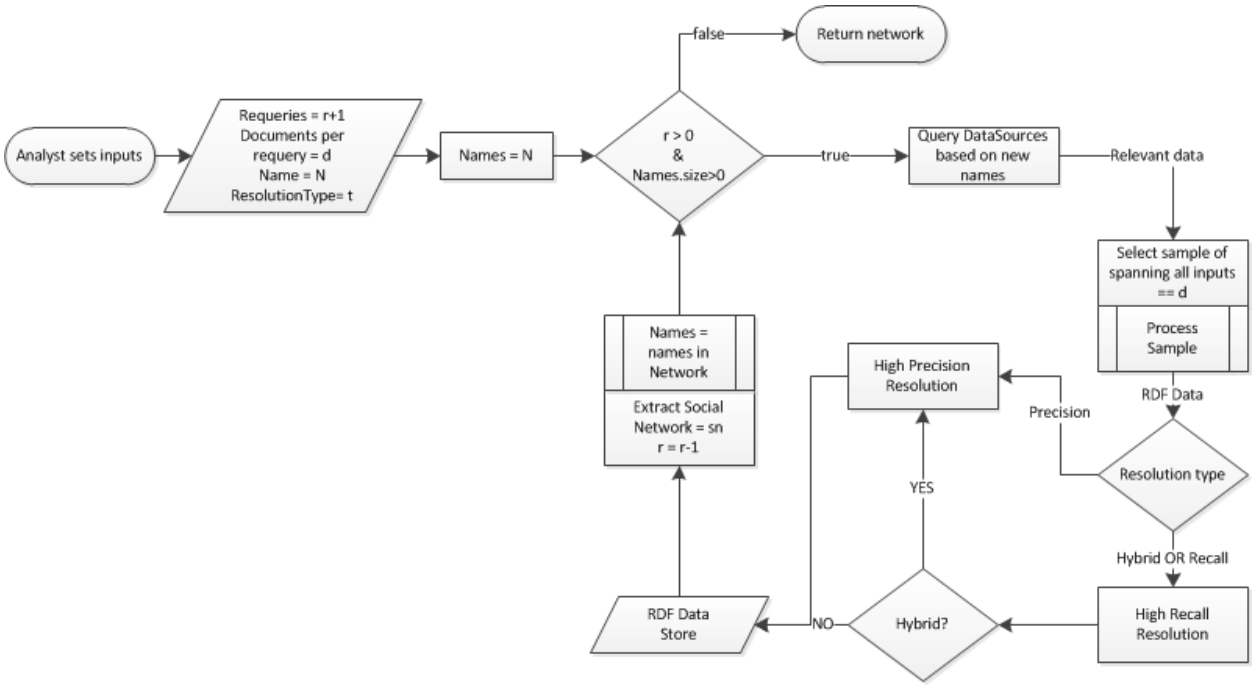

Figure 7. Social Network Extraction Process

For this algorithm, the user has the following parameters to select:

Name – This can be any known name or alias for a person of interest. The name should have at least two parts. The reason for this stipulation is the simple fact that only a subset of the data can be processed in a prudent manner. The more common the name, the more likely that incorrect irrelevant documents will be processed, thus resulting in undesirable networks. For example, consider the name “John Smith”.

Requeries – This parameter sets the number of times that the algorithm will explore the network and try to expand based on newly discovered people in the network. Since this where the bulk of the work is done, as this number increases, the algorithm will take longer to finish; however, this is not the only parameter that affects the running time.

Documents per requery – This is the second parameter which affects the running time. As this number increases, each iteration of the main loop will take a longer time to complete, however more data will have been processed, thus providing more data for the network to be discovered.

Resolution type – This selection allows for three different entity resolution options. The High Precision model is strictly based on the entity resolution model (medium speed). The High Recall option works strictly off of a name comparison. It merges people whose name parts (including white space) have a combined length of greater than 8 characters. This is so that in the event that there is not enough information about a two people to co-refer them together, they can

still be used to grow the network (fast speed). The Hybrid mode first runs High Recall entity resolution then High Precision entity resolution. This option was created because there are cases where data may not contain enough of the information required to merge people together, resulting in redundant nodes or disjoint graphs containing the person in question. When the Hybrid entity resolution option is selected, it allows the High Precision process to obtain richer data about the entities and further resolve persons which are missed by the High Recall section (slow speed). It is important to note that High Recall and Hybrid can result in improper sections of the network once a common name is introduced.

5 Summary

The vast amount of data currently available or available in the near future, provides the opportunity for innovative research in transportation safety. The era of 'Big Data' has begun to take hold in many areas of research. Key scientific breakthroughs have already come from the computational analysis of Big Data in the fields of genomic research, astronomy, macromolecular structure, and healthcare. The real value comes from the patterns and insights that can be obtained by making connections between, and extractions from, disparate, structured and unstructured datasets. Currently existing archival crash data, roadway characterization data, and recently collected SHRP 2 naturalistic driving data can already be considered sources of big data; however, the lack of linkages to each other and to other datasets limits their value. In the near future, even more data is expected to be available including near real time data to support transportation systems analyses, in general, and safety analyses in particular. This data is likely to come from sources such as connected vehicles and infrastructure sensors, next generation 911 crash reports, news media crash reports, and weather (i.e., mPing) and congestion information from social media or other streaming sensor data.

To take advantage of the vast amount of data available to transportation safety analysts now and in the near future, it is necessary to develop tools to handle and analyze data in an efficient manner. With that in mind, this proposed prototype system employs an informatics approach which considers how transportation safety datasets are best accessed and ingested, and how they can be rapidly processed using a distributed parallel architecture. This design will produce an analytic framework that looks to the future and provides system extensibility to facilitate the incorporation of new sources of data as they become available, while also considering future data mining opportunities.

The proposed prototype architecture has unique characteristics that address the formidable challenges of multisource data alignment and analytics. These characteristics include the alignment of multiple, disparate data sources via a common data model; the rapid integration of new data sources; mature entity and event coreference resolution capabilities; and advanced analytics such multi-source social network analysis and location normalization. A key feature to this prototype system that there is no need for a data warehouse, which might otherwise act as a common data model. Instead, a non-data-warehousing solution is built on cloud-enabled technologies. On demand, raw data is translated into a singular graph structure. The system in essence operates as a virtual fusion center, and can be installed on any cloud system. Cloud technologies such as Hadoop and Storm allow the system to take full advantage of the parallelism and scalability that is native to the cloud. The overall system architecture has been carefully designed so that it is compatible with the technologies currently employed in the state-of-the-art cloud environments.

References

[1] Apache. (2018). “Apache Hadoop.” (software) Wakefield, MA. Available online: http://hadoop.apache.org/, Last accessed September 2018.

[2] CentOS Project. (2018). “CentOS.” (software) Available online: https://www.centos.org/, Last accessed September 2018.

[3] Hortonworks. (2018). “Hortonworks HDP.” (software) Santa Clara, CA. Available online: https://hortonworks.com/products/data-platforms/hdp/, Last accessed September 2018.

[4] Apache. (2018). “Apache Ambari.” (software) Wakefield, MA. Available online: http://ambari.apache.org/, Last accessed September 2018.

[5] Talend. (2018). “Talend.” (software) Redwood City, CA. Available online: https://www.talend.com/, Last accessed September 2018.

[6] Hitachi Vantara (2018). “Pentaho Data Integration” (software) Santa Clara, CA. Available online: https://www.hitachivantara.com/en-us/products/big-data-integration-analytics/pentaho-data-integration.html/, Last accessed September 2018.

[7] Informatica. (2018). “Informatica.” (software) Redwood City, CA. Available online: https://www.informatica.com/#fbid=_Zqx2rW346I, Last accessed September 2018.

[8]D2RQ. (2018). “D2RQ.” (software) Berlin, Germany. Available online: http://d2rq.org/, Last accessed September 2018.

[9] Lymba Corp. (2018). “Lymba Pipeline” (software) Richardson, TX. Available online: https://www.lymba.com/Last accessed September 2018.

[10] Apache. (2018). “Apache Spark.” (software) Wakefield, MA. Available online: http://spark.apache.org/, Last accessed September 2018.

[11] Apache. (2018). “Apache Hive.” (software) Wakefield, MA. Available online: http://hive.apache.org/, Last accessed September 2018.

[12] Apache. (2018). “Apache Storm.” (software) Wakefield, MA. Available online: http://hive.apache.org/, Last accessed September 2018.

[13] Apache. (2018). “Apache Jena.” (software) Wakefield, MA. Available online: http://hive.apache.org/, Last accessed September 2018.