2016 IEEE Second International Conference on Big Data Computing Service and Applications

Spark-based Political Event Coding

Mohiuddin Solaimani*, Rajeevardhan Gopalan*, Latifur Khan*, Patrick T. Brandt†, Bhavani Thuraisingham*
* Department of Computer Science
The University of Texas at Dallas
Richardson, TX
{mxs121731, rxg132930, lkhan, bhavani.thuraisingham}@utdallas.edu

† School of Economic, Political, and Policy Sciences
The University of Texas at Dallas
Richardson, TX
pbrandt@utdallas.edu

***Abstract*—Political event data have been widely used to study international politics. Previously, natural text processing and event generation required a lot of human efforts. Today we have high computing infrastructure with advance NLP metadata to leverage those tiresome efforts. TABARI – an open source non distributed event-coding software – was an early effort to generate events from a large corpus. It uses a shallow parser to identify the political actors, but ignores semantics and relation among the sentences. PETRARCH, the successor of TABARI, encodes event data into "who-did-what-to-whom" format. It uses Stanford CoreNLP to parse sentences and a static CAMEO dictionary to encode the data. To build dynamic dictionaries, we need to analyze more metadata such as the token, Named Entity Recognition (NER), co-reference, and many more from parsed sentences. Although these tools can code modest amounts of source corpora into event data they are too slow and suffer scalability issues when we try to extract metadata from a single document. The situation gets worse for other languages like Spanish or Arabic. In this paper, we develop a novel distributed framework using Apache Spark, MongoDB, Stanford CoreNLP, and PETRARCH. It shows a distributed workflow by using Stanford CoreNLP to extract all the metadata (parse tree, tokens, lemma, etc.) from the news corpora of the Gigaword dataset and storing it to MongoDB. Then it uses PETRARCH to encode events from the metadata. The framework integrates both tools using distributed commodity hardware and reduces text processing time substantially with respect to a non-distributed architecture. We have chosen Spark over traditional distributed frameworks like MapReduce, Storm, Mahout. Spark has in-memory computation and lower processing time in both batch and stream processing compared to other options.**

***Index Terms*—Political event; TABARI; Stanford CoreNLP; PETRARCH; MongoDB; Apache Spark**

I. Introduction

Modern social and political conflicts typically emerge in geographically constrained regions before propagating through the larger international system. Such propagation occurs through both spatial proximity and complex inter-dependencies among actors and related phenomena. For instance, it is not uncommon for emerging threats to include elements of political conflict, criminal violence, and disease that are each geographically and causally interconnected. No social science dataset currently exists that provides accurate structured data on political and social events around the globe, with historical coverage, drawn from multiple language sources, and is freely available within hours of the events occurring.

Political event data are records of interactions among political actors using common codes for actors and actions, allowing for aggregate analysis of political behaviors. These include both material and verbal interactions between political entities, such as bombings, protests, or diplomatic consultations. These data are collected at the event level, the smallest aggregation possible, allowing for both low-level and aggregate analysis of political behaviors. As such, they enable us to quantify the propagation of an event or a series of events through space and time, while also capturing the complex interactions of the actors involved.

Human coding was the standard method to generate event data before the 1990s, and remains the gold standard for validating machine-coded output. Unfortunately, human capabilities have their limits as researchers cannot be counted on to do this task with sufficient speed and accuracy [1], [2]. Humans are capable of providing high-quality gold standard cases for reference purposes, but that is extremely laborious and cross-validated gold standard cases can only be generated at a rate of three to six events per hour, which cannot be applied to near-real time production-level event generation involving millions of records.

Human coders who are not working in a gold-standard environment - that is, they are simply trying to generate event data as efficiently as possible - appear to have roughly the same accuracy level as machine-coding systems (70% to 80%) and in some documented instances are far worse (25% and 40% is documented) [3]. Thus, for event data with daily updates, machine coding is the only option.

Developments in open source Natural Language Processing (NLP) software such as Stanford CoreNLP [4] [5], PETRARCH [6] allow for more accurate content extraction from each sentence and support text analysis in multiple languages, facilitating for the first time the analysis of foreign language texts without requiring time intensive word-by-word translation of each news story. These developments add computational complexities and hence increase the overall pro-

978-1-5090-2251-9/16 $31.00 © 2016 IEEE
DOI 10.1109/BigDataService.2016.30

IEEE computer society

cessing time of standalone applications. Due to this, standalone applications are not suitable when processing a huge corpus of data since they suffer from scalability issues. There is, hence, a demand for scalable frameworks capable of handling Big Data with low latency. Popular Big Data frameworks, e.g., Hadoop [7], MapReduce [8], HBase [9], Mahout [10], Google Bigtable [11], MongoDB [12] etc. are highly scalable and more geared towards batch processing. There are some frameworks that have stream processing capabilities like Apache Storm [13], Apache S4 [14] and Apache Spark [15], [16]. Among these, Spark performs best for both batch and stream processing [15], [16]. Spark performs in-memory analytics and runs a streaming computation as a series of micro batch jobs. It makes the batch size as small as possible to achieve lower end latency. It maintains the states between batches in-memory so that it can recover quickly. Spark has an advanced DAG (Direct Acyclic Graph) execution engine that supports cyclic data flow and in-memory computing. This makes Spark faster than other distributed frameworks. Spark is 100 times faster than Hadoop MapReduce in-memory, or 10 times faster on disk [15]. It is also faster than Storm [17] and S4 [18]. Overall, it generates low latency, real-time results. It has instant productivity and no hidden obstacles. It also has an easy cluster setup procedure.

To address these scalability issues in political event coding using existing applications, we have come up with a distributed framework using Apache Spark which comprises of two workflows. The first workflow extracts all the metadata information from the documents using Stanford CoreNLP and stores them in MongoDB. The reason for storing all the metadata as-is in the MongoDB is that even though PETRARCH needs only the sentences and their parse trees, the extra metadata information can be used to dynamically add actors to the CAMEO dictionary. The second workflow uses the extracted metadata information to generate political event code data using PETRARCH. By running these two workflows inside Apache Spark worker jobs, we are able to achieve near real-time [19] political event coding for large corpus of data.

In this paper, we have the following contributions:

- We develop a novel distributed framework using Apache Spark, MongoDB, Stanford CoreNLP and PETRARCH.
- We show a distributed work flow that extracts all the metadata (parse tree, tokens, lemma, etc.) from the news source, stores it to MongoDB, and later generates events.
- We compare the processing time of distributed versus non-distributed frameworks with a set of experiments.

The rest of the paper is organized as follows: Section II gives a background on the various open source tools and concepts used in this paper. Section III talks about the non-distributed version of political event coding. Section IV explains how we have implemented the political event coding in a distributed manner using Apache Spark. Section V shows the experimental result. Section VI covers the related works. Section VII states the conclusion and future works. Section VIII is the acknowledgment followed by bibliography.

II. Background

The following concepts are pertinent to our framework:

A. Apache Spark

Apache Spark [15] is an open-source distributed framework for data analytics. It avoids the I/O bottleneck of the conventional two-stage MapReduce programs. It provides the in-memory cluster computing that allows a user to load data into a cluster's memory and query it efficiently. This increases its performance faster than Hadoop MapReduce [15]. Figure 1 shows the Spark architecture. It has a driver node called Master and slave nodes called Worker. Workers can communicate with HDFS with Hadoop's Data nodes if any I/O operations are required.

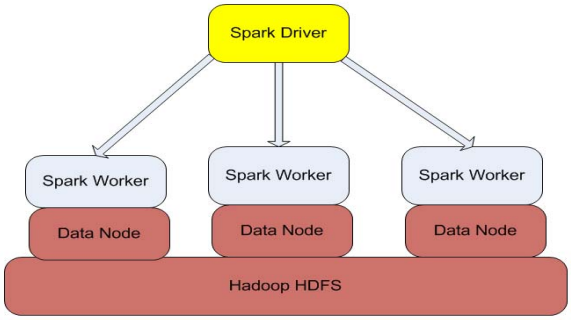

Fig. 1. Spark architecture

Spark has two key concepts: Resilient Distributed Dataset (RDD) and Directed Acyclic Graph (DAG) execution engine.

- *Resilient Distributed Dataset (RDD)*
  RDD [20] is a distributed memory abstraction. It allows in-memory computation on large distributed clusters with high fault-tolerance. Spark has two types of RDDs: parallelized collections that are based on existing programming collections (like list, map, etc.) and files stored on HDFS. RDD performs two kinds of operations: transformations and actions. Transformations create new datasets from the input or existing RDD (e.g. map or filter), and actions return a value after executing calculations on the datasets (e.g. reduce, collect, count, saveAsTextFile, etc.).
- *Directed Acyclic Graph (DAG) execution engine*
  Whenever the user runs an action on RDD, a directed acyclic graph is generated considering all the transformation dependencies.

Spark supports both batch process and also processing of streaming data [21]. Its streaming component is highly scalable, and fault-tolerant. It uses a micro-batch technique which divides the input stream as a sequence of small batched chunks of data for a small time interval. It then delivers these small packed chunks of data to batch system for processing.

Spark Streaming has two types of operators:

- *Transformation operator*
  It creates a new DStream [18] from one or more parent streams. It can be either stateless (independent on each interval) or stateful (share data across intervals).

- *Output operator*
  It is an action operator, and allows the program to write data to external systems (e.g., save or print DStream).

Like MapReduce, map is a transformation function that takes each dataset element and returns a new RDD. On the other hand, reduce is an action function that aggregates all the elements of the RDD and returns the final result.

B. MongoDB

MongoDB [12] is an open source NOSQL document database. MongoDB stores data in the form of a BSON (Binary JSON-like) document. This makes its integration of data in certain types of applications easier and faster. MongoDB has key with a complex data structure called as a document. A set of documents forms a collection.

C. Stanford CoreNLP

Stanford CoreNLP [5] is an useful framework for annotating text. Stanford CoreNLP includes tools like part-of-speech (POS) tagger, named entity recognizer (NER), parser, coreference resolution system, sentiment analysis, and bootstrapped pattern learning tools. It supports models for other languages. It has natural language analysis tool embedded in it and takes raw text as input and gives the tokens, Parts-of-Speech, whether they are names of companies, people, etc., normalize dates, times, and numeric quantities, and marks up the structure of sentences in terms of phrases and word dependencies, indicates which noun phrases refer to the same entities, indicates sentiment, etc. It has flexible and extensible interface and helps to ease the application of linguistic analysis tools from text. It is also simple and easy to use.

CoreNLP is formed by two classes: Annotation and Annotator. Annotations are the data structures that hold the results of Annotators. Annotations are maps, e.g., the parse, the part-of-speech tags, or named entity tags. Annotators are functions and operate over Annotations, e.g., tokenize, parse, or NER tag sentences. Both Annotators and Annotations are glued by Annotation Pipelines to create generic Annotators.

D. TABARI

Textual Analysis By Augmented Replacement Instructions (TABARI) [22] – is an open-source program. It is a successor to KEDS (Kansas Event Data System) [23]. TABARI extracts event data from the stream of text using sparse pattern recognition techniques [24]. It generates event data as a semi-formalized structure like (date, source, target, event). It uses shallow parsing procedures which identifies the constituents of sentences. However, it does not specify the internal semantics and their role in the sentence like Stanford CoreNLP.

E. PETRARCH

PETRARCH (A Python Engine for Text Resolution And Related Coding Hierarchy) [6] is a natural language and processing tool to produce machine-coding events data. It takes fully-parsed text summaries in Penn Tree format [25] from Stanford CoreNLP and generates event like 'who-did-what-to-whom' format. PETRARCH uses CoreNLP to parse sentences and generate the events. It is a successor to TABARI and handles the noun/verb/adjective disambiguation that accounts for much of the size of the TABARI dictionaries. It uses synonym sets from WordNet[26] to identify actors even if they are not in the dictionaries.

III. Political event coding

Advances in NLP improve the fidelity of machine-coded events and can be done in near real time with high accuracy. Coding open-source news articles into structured text, analysts can create rich baselines and forecasts of fast-moving political events. Developments in Stanford's CoreNLP software allow for more accurate content extraction from each sentence and support text analysis in multiple languages, facilitating the analysis of foreign language texts, and also the geolocation of events. The language extensions flow from those coded in Stanford CoreNLP. While PETRARCH was developed using English language texts, the coding engine relies exclusively on the Penn TreeBank tags that are produced in the CoreNLP markup, and these are standardized across languages.

The existing event coding software, PETRARCH, requires hand-crafted actor dictionaries for coding the subject and objects of the actions or statements from news reports. Manually creating these dictionaries is a laborious task. We therefore need to automatically expand actor dictionaries using NLP techniques. Our goal for automatic expansion of actor dictionaries takes as input (1) a set of existing actor dictionaries that are to be expanded, (2) the event dictionaries employed by PETRARCH, and (3) a large raw corpus containing the documents of interest. From the event dictionaries, we first extract all the event words of interest. Then, we identify all occurrences of these event words from the raw corpus. After that, we use the Stanford CoreNLP tool to identify the subject and object of each of these event mentions. These noun phrases are the actors that we will use to expand the actor dictionaries. To increase coverage, we therefore propose to train a sequence model for extracting additional actors from the corpus. Modifying PETRARCH to identify new actors should be primarily an issue of dictionary development. Stanford CoreNLP (and thus PETRARCH) can employ fully parsed sentences with other metadata information (NER, lemma, etc.) to build this dynamic dictionary.

Figure 2 shows the political event coding flow. It starts with collecting the political news stories. Open source web scraper (e.g., Goose [27]) allows collecting arbitrary news pages. Instead of collecting arbitrary news pages, a white list of RSS (Rich Site Summary) [28] feeds is utilized for gathering relevant news stories.

The news stories are in raw format and cannot be fed into standard parser. It requires filtering and preprocessing to convert into structured data. We make use of the Gigaword dataset [29] which is a collection of structured news data. It is SGML (Standard Generalized Markup Language) format and has collection of news story from multiple source languages.

The next crucial part is to extract metadata from the structured news stories. Stanford CoreNLP allows us to parse

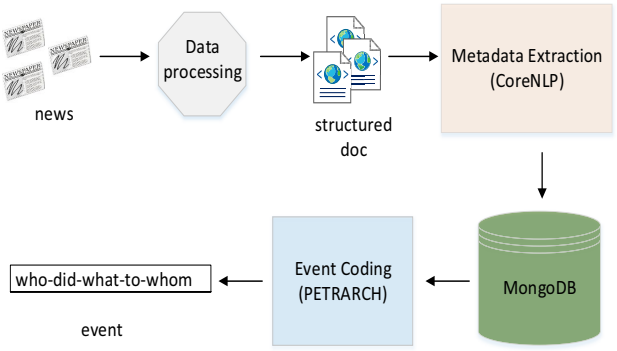

Fig. 2. Political event coding

each sentence and collect metadata accurately. It gives us Part-of-Speech, tokens, lemma (canonical form of a set of words), NER (Name Entity Recognition), parse tree, dependencies, co-reference, sentiment, and so on. These metadata are extremely important to improve dictionaries.

In the following example, we have shown how to generate metadata using CoreNLP. Figure 3 visualizes the Part-of-Speech tagging and NER of the first sentence from the paragraph. Figure 4 has the dependency between them. Figure 5 shows the co-reference of the four sentences of the paragraph. Besides that, we also show the token, lemma, parse tree (Penn Tree Bank format) and sentiment of the first sentence of the paragraph.

The event data generated from PETRARCH is of the form 'who-did-what-to-whom'. It outputs the Source, the Target and a CAMEO code [30] for the political event along with other information like the date of event, news source, etc.

"Officials said that, in response to the attacks in Paris, the administration was seeking renewed global commitment to that intensified military action, and to a negotiated settlement of Syria's civil war.

France's retaliation came as Obama held talks with allied leaders and with Russian President Vladimir Putin at the summit being held in this Turkish Mediterranean resort city.

Obama vowed again on Sunday to help France hunt down the perpetrators of the attacks." [31] - The Washington Post 11/15/2015

The Part-of-Speech tags used in the following examples will be found in [32].

```
Sentence

"Officials said that, in response to the attacks in
Paris, the administration was seeking renewed global
commitment to that intensified military action, and
to a negotiated settlement of Syria's civil war."
```

```
Token

Officials, said, that, in, response, to, the,
attacks, in, Paris, the, administration, was,
seeking, renewed, global, commitment, to, that,
intensified, military, action, and, to, a,
negotiated, settlement, of, Syria,'s, civil,
war,.

Lemma

official, say, that, in, response, to, the,
attack, in, Paris, the, administration, be,
seek, renew, global, commitment, to, that,
intensify, military, action, and, to, a,
negotiate, settlement, of, Syria, 's, civil,
war,.

Parse

(ROOT
 (S
  (NP (NNS Officials))
  (VP (VBD said)
   (SBAR (IN that)
    (S (, ,)
     (PP (IN in)
      (NP
       (NP (NN response))
       (PP (TO to)
        (NP
         (NP (DT the) (NNS attacks))
         (PP (IN in)
          (NP (NNP Paris)))))))
     (, ,)
     (NP (DT the) (NN administration))
     (VP (VBD was)
      (VP (VBG seeking)
       (NP
        (NP (VBN renewed) (JJ global) (NN commitment))
        (PP
         (PP (TO to)
          (NP(DT that)(JJ intensified)(JJ military)(NN action))
          (, ,)
          (CC and)
          (PP (TO to)
           (NP
            (NP (DT a) (VBN negotiated) (NN settlement))
            (PP (IN of)
             (NP
              (NP (NNP Syria) (POS 's))
              (JJ civil) (NN war)))))))))))
  (. .)))

Sentiment

1

Event(PETRARCH)

  Date    Source  Target  CAMEO   DOC ID       News Source
20151115  FRA     USAGOV   040  WP20151115_1       WP

FRA = France
USAGOV = US Government (Obama)
040 = Cooperate
WP = The Washington Post
```

MongoDB stores all the metadata and later they are fed as input to PETRARCH to generate events. MongoDB is a NoSQL database and stores all the data in BSON (Bi-

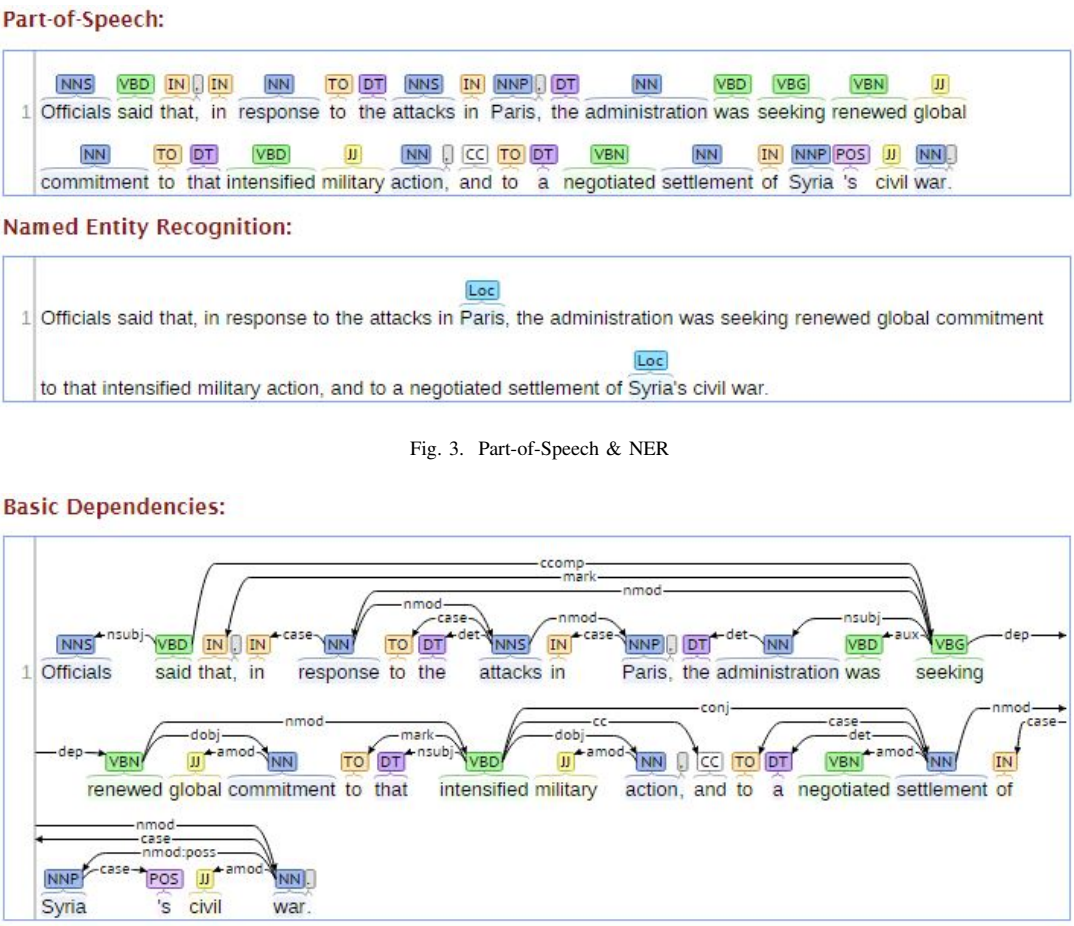

Fig. 3. Part-of-Speech & NER

Fig. 4. Basic Dependency

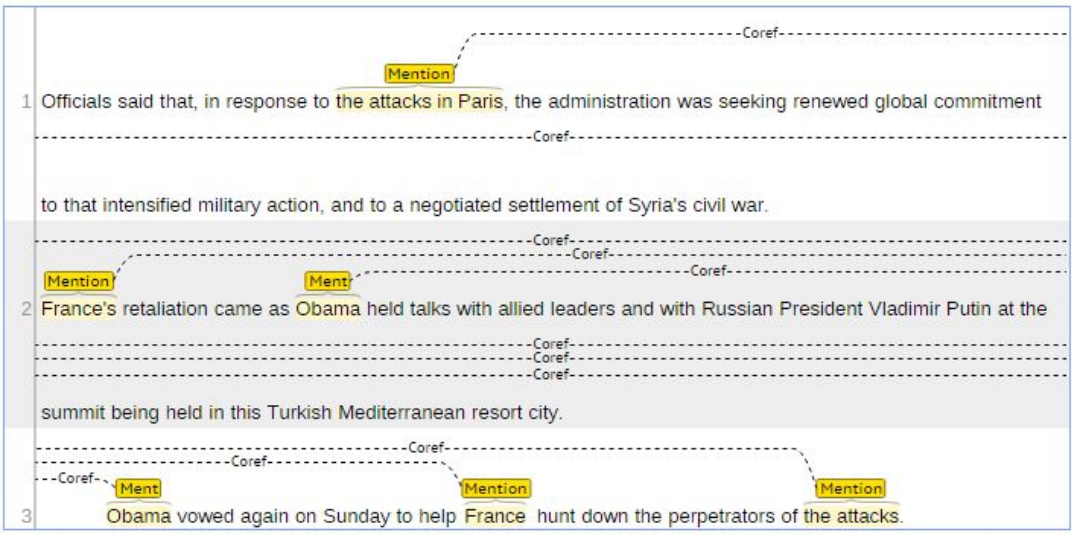

Fig. 5. Co-reference

nary JSON) format. PETRARCH uses CAMEO dictionary to identify political events and discards other events from the news data. PETRARCH gives the output as 'who-did-what-to-whom' format and it is what we refer to as event data. More specifically, event data consists of three components, {SOURCE_ACTOR, ACTION_TYPE, TARGET_ACTOR} as well as general attributes {DATE_TIME, LOCATION}.

A sample DB record is given bellow.

```
MongoDB record

{
type: story,
doc_id: WP_ENG_20151115.0017R,
head_line: France launches..,
date_line: ANTALYA, Turkey, Nov 15
sentences: [
        {
         sentence_id: 1,
          sentence: Officials...,
          parse_sentence: .. ,
          dependency_tree: ..,
          token: ..,
          lemma: ..,
          ner:   ..,
          coref: ..,
          sentiment: ..
        }

     ]
}
```

IV. Spark-based political event coding

Stanford CoreNLP and PETRARCH are used to code political event. These tools perform accurately but suffer from slow processing time when run on huge news corpus. A significant bottleneck comes from the Stanford CoreNLP because each annotator needs sufficient amount of time to process a document. Adding more annotators makes it much slower. As a result, a single document with a couple of sentences requires more than a few minutes, so processing large number of news stories in real time is a challenge. Also PETRARCH is slower than its successor TABARI [33]. Considering this, we have no choice but to develop a distributed framework that will integrate both CoreNLP and PETRARCH in a parallel manner and perform the event coding in close to real time.

Motivated by thess ideas, we have designed a distributed framework with Apache Spark. Spark is our obvious choice over existing distributed frameworks like MapReduce [7], Storm [13], etc. because of its in-memory computation and lower execution time [21], [17]. In our two part framework, we show a work flow for extracting metadata from news stories using CoreNLP and also coding event from them using PETRARCH. First, it has a distributed program that executes CoreNLP in each Spark's worker nodes in parallel to extract metadata from news story and stores it to MongoDB. Second, it runs another distributed programs that executes PETRARCH in parallel inside each worker node to generate political events from data stored in MongoDB. Figure 6 describes our Spark-based political event coding framework. We will discuss it thoroughly in the following subsections.

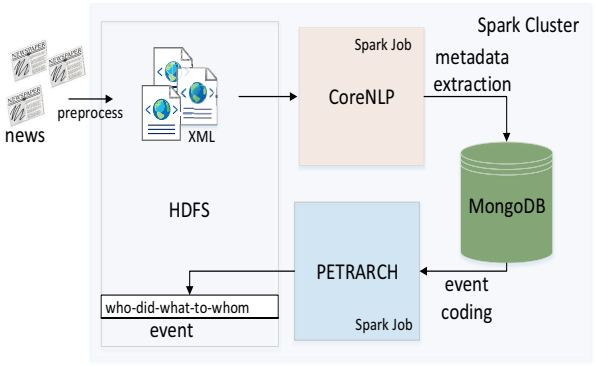

Fig. 6. Spark-based political event coding framework

A. News story collection and preprocess

News stories are collected using web scrappers, filtered, and stored in structured format (SGML). Each SGML file consists of multiple thousands of news documents. For example, we have following sample file in Gigaword dataset.

```
<DOC id="WP_ENG_20151115.0017R" type="story" >
<HEADLINE>France launches..</HEADLINE>
<DATELINE>ANTALYA, Turkey, Nov 15</DATELINE>
<TEXT>
<P>Officials said that, in response to the attacks in
Paris, the administration was seeking renewed global
commitment to that intensified military action, and
to a negotiated settlement of Syria's civil war.
</P>
<P>
...
</P>
</TEXT>
</DOC>
<DOC>
...
</DOC>
```

These documents require further processing before feeding them into our framework, after which we store each file in HDFS (Hadoop Distributed File System) where each line of the file contains single document.

```
<DOC> ... <DOC>
<DOC> ... <DOC>
...
```

B. Metadata extraction

We are using CoreNLP to parse sentences and generate all the metadata for a document. After that, we store the extracted metadata in MongoDB. We have run the CoreNLP parsing inside each worker in the Spark job. Algorithm 1, ExtractMetaData describes the procedure for metadata extraction. We read all the files from HDFS and store them in RDD (line 2). The RDD now contains all the documents. We apply the mapping on RDD. The map function (line 3-5) is run on worker node in parallel. Inside map, we process

each document. For each document, we parse the sentences with the required annotator using CoreNLP and obtain a list of metadata information (line 4). Once metadata are obtained, a MongoDB object for that document is created using that list (line 5). So, the map function gives us the list of MongoDB objects. Each of them is a JSON-like object that contains all the metadata for that document. After getting the object list, SAVERDDTOMONGODB stores all DB objects to MongoDB (line 6). We have used the Hadoop-based MongoDB connector API [34] to make the Spark cluster communicate with MongoDB. It has a function (SAVEASNEWAPIHADOOPFILE) that stores data to MongoDB in parallel like reducer in MapReduce.

Algorithm 1 Extract Metadata

1: **procedure** EXTRACTMETADATA
2: $hrdd \leftarrow hadoopRDD.readFiles$
3: $hrdd.map\{$
4: $list \leftarrow Generate\ metadata\ using\ CoreNLP$
5: $MongoDBObject.create(list)$
6: $\}.saveRDDtoMongoDB()$

C. Event coding

MongoDB stores all the information to code events. PETRARCH generates the political event codes. For each document in the MongoDB, it is encoded by passing it as input to PETRARCH, which is run inside a Spark worker and political event codes are generated. Algorithm 2, CODINGEVENT describes the overall process. We use the Hadoop MongoDB connector to read documents from MongoDB (line 2). These documents are read in parallel and stored in RDD. We apply the mapping on RDD. The map functions are run in parallel in each worker where we use PETRARCH to code the events. Inside the map function (line 3-4), we pass the documents with their sentences and parse trees to the PETRARCH and generate the political event codes. The generated political events are stored (like reducer in MapReduce) in a file in HDFS (line 5).

Algorithm 2 Coding Event

1: **procedure** CODINGEVENT
2: $mongordd \leftarrow mongoRDD.readFromMongoDB$
3: $mongordd.map\{$
4: $events \leftarrow Generate\ events\ using\ PETRARCH$
5: $\}.saveRDDtoHDFS()$

D. Design challenges

Parsing with CoreNLP is a time consuming and memory intensive job. It occupies almost the entire memory of a Spark worker node when processing a large corpus of text. As a result, the Spark jobs are frequently halted when the Spark's heap memory exceeds its allocated limit. An immediate mitigation would be to increase the worker node heap memory when this happens.

V. EXPERIMENTAL RESULT

A. Dataset

We have chosen English Gigaword dataset [29] for our experiment. It has news stories from six different news agencies:

- Agence France-Presse, English Service (AFP)
- Associated Press Worldstream, English Service (APW)
- Central News Agency of Taiwan, English Service (CNA)
- Los Angeles Times/Washington Post Newswire Service (LTW)
- New York Times Newswire Service (NYT)
- Xinhua News Agency, English Service (XIN)

'It has total 722 files with more than 7 million documents. Each document contains document id, head line, date line and news type and news story. The news comes from multiple news type like story, multi (summary of story), advis (advisory), and other. In our experiment, we select documents from AFP and CNA and consider only story type 'news'. Datta

B. Cluster setup

Our VMware cluster consists of 5 VMware ESXi [35] (VMware hypervisor server) 5.5 systems. Each of the systems has Intel(R) Xeon(R) CPU E5-2695 v2 2.40GHz processor, 64 GB DDR3 RAM, 4 TB hard disk and dual NIC card. Each processor has 2 sockets and every socket has 12 cores. So there are 24 logical processors in total. All of the ESXi systems contain 3 virtual machines. Each of the virtual machines is configured with 8 vCPU, 16 GB DDR3 RAM and 1 TB Hard disk. As all the VM's are sharing the resources, performance may vary in run time. We have installed Linux Centos v6.5 64 bit OS in each of the VM along with the JDK/JRE v1.8. We have installed Apache Spark version 1.5.0. We have also installed Apache Hadoop NextGen MapReduce (YARN) [36] with version Hadoop v2.6.0 and formed a cluster. Table I has the summary of the Spark cluster.

TABLE I
SPARK CLUSTER SUMMARY

Total Master node	1
Total slave node	12
vCPU (core) in each node	8
Memory in each node	16GB
Storage in each node	1TB

C. Result

As parsing with CoreNLP is time consuming, it is not feasible to conduct a non-distributed experiment over all the documents. However, it is also important to compare the execution time of non-distributed and distributed frameworks. That is why, we have two distinct sets of experiments. First, we are comparing the two frameworks. We have chosen a document set which has 12,484 documents of story type news for this experiment. Second, we run our distributed framework for CNA and AFP documents and try to compare with the non-distributed with estimation.

Table II shows the comparison of document processing times using CoreNLP. It shows also scale-up by using following equation.

$$Scale\ up = \frac{Non\ distributed - Distributed}{Non\ distributed} \times 100 \quad (1)$$

We know that each annotator of CoreNLP takes extra time to generate metadata, so we list them individually and show the time comparison between distributed and non-distributed parsing. Table II shows us that the distributed framework requires less time than non-distributed framework for each annotator. Furthermore, if we consider all of the annotator, then distributed framework only requires 15.34 minute to process 12,484 documents whereas a non-distributed framework needs 17.60 hour.

TABLE II
ANNOTATOR PROCESSING TIME BY CORENLP (12,484 DOCUMENTS)

Annotator	Distributed	Non-distributed	Scale up (%)
token	**51.96s**	08.03m	**89.00**
lemma	**60.14s**	08.60m	**88.00**
ner	**11.76m**	10.12h	**98.00**
parse	**09.71m**	07.82h	**97.00**
dependency	**10.45m**	8.00h	**98.00**
dcoref	**14.72m**	17.33h	**98.00**
sentiment	**10.08m**	07.80h	**97.00**
All of the above	**15.34m**	17.60h	**98.00**

Table III lists the document statistics when 12,484 documents are used for political event generation using PETRARCH. Out of 12,484 documents, only 2,127 documents are considered for political event generation and the rest are discarded. This is expected because PETRARCH only encodes political events and discards any other news events. A total of 2,134 political event codes are generated from these 2,127 documents.

Table IV compares the distributed framework for event coding with the non distributed one. The distributed framework requires only 13.22 milliseconds of processing time where as the non-distributed frameworks takes 1.26 minutes.

TABLE III
EVENT CODING USING PETRARCH

Skipped documents	1,556
No event documents	8,801
Event documents	2,127
Total documents	**12,484**
Total events	**2,134**

Table V shows the CoreNLP processing time of the distributed framework versus its non-distributed counterpart. Here we consider all the annotators listed in Table II. We have conducted our experiment on CNA and AFP news. In both cases, we have estimated time for non-distributed framework. CNA has 95,777 documents and our distributed framework takes only 2.9 hours to process all the documents where as the estimated time of a non-distributed framework is 16 days. In the same way AFP's 0.9 million records require only 17.37 hours, whereas the non-distributed framework is estimated to run for a couple of months.

TABLE IV
PROCESSING TIME BY PETRARCH

Total events	Distributed	Non-distributed	Scale up (%)
2,134	**13.22ms**	1.26m	**99.00**

Similarly, we show that the event coding time comparison for AFP and CNA news documents. We can see from table VI, that CNA has 16541 events and takes 102.47 milliseconds for event code generation, and AFP has 14,8738 events and takes 921.42 milliseconds in a distributed framework whereas in a non-distributed framework it is estimated to take couple of minutes.

TABLE V
TOTAL PROCESSING TIME BY CORENLP

	Total documents	Distributed	Non-distributed (Estimated)
CNA	95,777	**02.19h**	16.00d
AFP	9,00,000	**17.371h**	5.20mo

TABLE VI
TOTAL PROCESSING TIME BY PETRARCH

	Total events	Distributed	Non-distributed (Estimated)
CNA	16,541	**102.47ms**	9.76m
AFP	1,48,738	**921.42ms**	87.82m

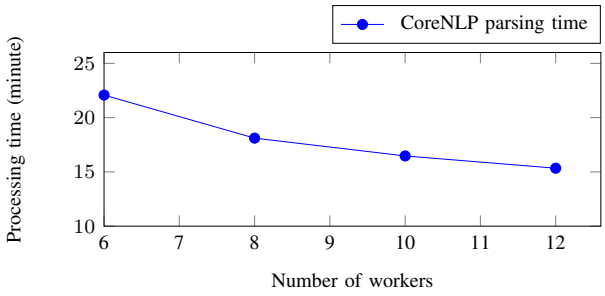

Fig. 7. CoreNLP parsing time over number of workers

Figure 7 illustrates the variation in the processing time of CoreNLP with the number of workers in Spark. Adding more workers reduces the parsing time significantly.

VI. RELATED WORK

Our related work covers political event coding and Big Data frameworks.

Political event data analysis is being carried out over many decades. Azar, et al. [37] in their COPDAB project have used political event data to study foreign policy. Baum, et al. [38] have studied political event data related to armed conflict.

Hammond, et al. [39] have described micro level study of political violence in their Global Database of Event Language and Tone (GDELT) while forecasting political conflict has been done recently by Chadefaux [40]

Political event data has been used to improve CAMEO dictionary. Thomas [41] scales up CAMEO dictionary to analyze categories of conflict and cooperation. Best, et al. [42] analyze TABARI coding system to identify actors and their relationship inside event data.

Very few frameworks exist to code political event. Schrodt and Van Brackle [43] shows a complete picture of generating events from raw news text. It describes the life cycle of event generation, key preprocessing steps and also real-time coding with existing NLP tools. However, it does not address the scalability issue. Schrodt and Van Brackle[33] also shows a framework EL:DIABLO that consists of both CoreNLP, and PETRARCH. It shows the comparison between TABARI and PETRARCH. It has the same scalability problem which is not mentioned previously.

Over the past few years, distributed frameworks have become more popular. They scale up with existing commodity hardware and handle large volume of data with lower processing time. We can divide out Big Data framework into two categories: streaming [15], [20] and non-streaming [36]. Streaming framework processes continuous data periodically. It may use machine learning techniques to analyze stream data. Spark, Storm, S4 are the key candidates to build up streaming framework. Mohiuddin, et al. [17], [44] have proposed distributed real-time anomaly detection frameworks using both Spark and Storm for VMware-based data center and also compare their processing time. Non-streaming Big data frameworks are built with Hadoop, MapReduce, HBase. They perform batch processing. Now a days, Spark becomes popular and replaces MapReduce. Nesi, et al. [45] show a distributed framework to search key phrases from web. They use MapReduce to build their framework. Al-Naami, et al [46] have developed a distributed Geographic Information System Querying Framework (GISQF) to process Massive Spatial Data. They have used SpatialHadoop [47] (extension of Hadoop) and MapReduce in their framework. Moreover, they have coded events from news text using TABARI and CAMEO.

MapReduce based frameworks are becoming obsolete and have been replaced with Spark, Storm, etc. Both Spark and Storm has lower processing time but Spark is the best for both batch and stream processing [21]. Considering all of these, we have built a framework with Spark, CoreNLP, and PETRARCH that can generate political events form the raw news text with lower processing time.

VII. Conclusion and Future work

In this paper, we have presented a novel distributed framework for political event coding. Our framework has a complete work flow of extracting metadata using CoreNLP, storing them into MongoDB, and feeding them to PETRARCH to generate event. All of these are run in parallel and our framework reduces the time substantially.

In the future, we will improve our framework to code event data in several non-English languages. Moreover, we can use the stored metadata to automatically update actor dictionary using novel class detection technique. Furthermore, we can extend our work to identify news location using machine learning techniques.

VIII. Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. SBE-SMA-1539302. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

This material is also based upon work supported by the National Science Foundation under Award No. CNS1229652 and the Air Force Office of Scientific Research under Award No. FA-9550-12-1-0077.

References

[1] M. M. Masud, C. Woolam, J. Gao, L. Khan, J. Han, K. W. Hamlen, and N. C. Oza, "Facing the reality of data stream classification: coping with scarcity of labeled data," *Knowledge and information systems*, vol. 33, no. 1, pp. 213–244, 2012.

[2] M. M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham, "A practical approach to classify evolving data streams: Training with limited amount of labeled data," in *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE, 2008, pp. 929–934.

[3] K. Eck, "In data we trust? a comparison of UCDP, GED, and ACLED conflict events datasets," *Cooperation and Conflict*, vol. 47, no. 1, pp. 124–141, 2012.

[4] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2014, pp. 55–60. [Online]. Available: http://www.aclweb.org/anthology/P/P14/P14-5010

[5] Stanford CoreNLP. Stanford CoreNLP. [Online]. Available: http://nlp.stanford.edu/software/corenlp.shtml

[6] PETRARCH. Petrarch. [Online]. Available: http://petrarch.readthedocs.org/en/latest/

[7] Apache Hadoop. [Online]. Available: http://hadoop.apache.org/

[8] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[9] Apache HBase. [Online]. Available: https://hbase.apache.org/

[10] Apache Mahout. [Online]. Available: https://mahout.apache.org/

[11] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[12] MongoDB. Mongodb. [Online]. Available: https://www.mongodb.com/

[13] Storm - distributed and fault-tolerant real-time computation. [Online]. Available: http://storm.incubator.apache.org/

[14] S4. [Online]. Available: http://incubator.apache.org/s4

[15] Apache Spark. [Online]. Available: http://spark.apache.org/

[16] Apache Spark. [Online]. Available: http://spark.apache.org/streaming/

[17] M. Solaimani, M. Iftekhar, L. Khan, B. Thuraisingham, and J. Ingram, "Spark-based anomaly detection over multi-source VMware performance data in real-time," 2014.

[18] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: A fault-tolerant model for scalable stream processing," Tech. Rep., 2012.

[19] J. Beieler, P. T. Brandt, A. Halterman, P. A. Schrodt, and E. M. Simpson, "Generating political event data in near real time: Opportunities and challenges," *Forthcoming of Computational Social Science: Discovery and Prediction. ed. by R. Michael Alvarez, Cambridge, Cambridge University Press*, pp. 98–120.

[20] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing."

[21] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. Mccauley, M. Franklin, S. Shenker, and I. Stoica, "Fast and interactive analytics over Hadoop data with Spark."

[22] TABARI. TABARI program and source code. [Online]. Available: http://eventdata.parusanalytics.com/software.dir/tabari.html

[23] D. J. Gerner and P. A. Schrodt, "The Kansas event data system: a beginner's guide with an application to the study of media fatigue in the Palestinian intifada," 1996.

[24] C. Friedman, L. Shagina, Y. Lussier, and G. Hripcsak, "Automated encoding of clinical documents based on natural language processing," *Journal of the American Medical Informatics Association*, vol. 11, no. 5, pp. 392–402, 2004.

[25] Penn Treebank. The Penn Treebank Project. [Online]. Available: https://www.cis.upenn.edu/~treebank/

[26] WordNet. Wordnet. [Online]. Available: https://wordnet.princeton.edu/

[27] GOOSE. Python-goose - article extractor. [Online]. Available: https://github.com/grangier/python-goose

[28] RSS. RSS. [Online]. Available: https://en.wikipedia.org/wiki/RSS

[29] Gigaword English. Gigaword English. [Online]. Available: https://catalog.ldc.upenn.edu/docs/LDC2009T13/

[30] CAMEO. Conflict and Mediation Event Observations (CAMEO) codebook. [Online]. Available: http://eventdata.parusanalytics.com/data.dir/cameo.html

[31] T. W. Post. France launches fierce assault on isis targets in syria. [Online]. Available: https://www.washingtonpost.com/politics/obama-vows-to-help-france-hunt-down-perpetrators-of-paris-terror-attacks/2015/11/15/a4628faa-8b74-11e5-9a07-453018f9a0ec_story.html

[32] Penn Treebank Project. Alphabetical list of part-of-speech tags used in the Penn Treebank Project. [Online]. Available: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

[33] P. A. Schrodt, J. Beieler, and M. Idris, "Threes a charm?: Open event data coding with EL:DIABLO, PETRARCH, and the Open Event Data Alliance."

[34] Github:mongo-hadoop. [Online]. Available: https://github.com/mongodb/mongo-hadoop

[35] VMware. vSphere ESX and ESXi Info Center. [Online]. Available: http://www.vmware.com/products/esxi-and-esx/overview

[36] Apache Hadoop NextGen MapReduce (YARN). [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[37] E. E. Azar, "The conflict and peace data bank (COPDAB) project," *Journal of Conflict Resolution*, vol. 24, no. 1, pp. 143–152, 1980.

[38] M. A. Baum and Y. M. Zhukov, "Filtering revolution reporting bias in international newspaper coverage of the Libyan civil war," *Journal of Peace Research*, p. 0022343314554791, 2015.

[39] J. Hammond and N. B. Weidmann, "Using machine-coded event data for the micro-level study of political violence," *Research & Politics*, vol. 1, no. 2, p. 2053168014539924, 2014.

[40] T. Chadefaux, "Early warning signals for war in the news," *Journal of Peace Research*, vol. 51, no. 1, pp. 5–18, 2014.

[41] G. D. Thomas, "Scaling CAMEO: Psychophysical magnitude scaling of conflict and cooperation," *Foreign Policy Analysis*, vol. 11, no. 1, pp. 69–84, 2015.

[42] R. H. Best, C. Carpino, and M. J. Crescenzi, "An analysis of the TABARI coding system," *Conflict Management and Peace Science*, vol. 30, no. 4, pp. 335–348, 2013.

[43] P. A. Schrodt and D. Van Brackle, "Automated coding of political event data," in *Handbook of Computational Approaches to Counterterrorism*. Springer, 2013, pp. 23–49.

[44] M. Solaimani, L. Khan, and B. Thuraisingham, "Real-time anomaly detection over VMware performance data using storm," 2014.

[45] P. Nesi, G. Pantaleo, and G. Sanesi, "A distributed framework for NLP-based keyword and keyphrase extraction from web pages and documents."

[46] K. M. Al-Naami, S. Seker, and L. Khan, "GISQF: An efficient spatial query processing system," in *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*. IEEE, 2014, pp. 681–688.

[47] A. Eldawy, Y. Li, M. F. Mokbel, and R. Janardan, "CG_Hadoop: computational geometry in MapReduce," in *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2013, pp. 294–303.