International Journal of Geographical Information Science
Vol. 00, No. 00, Month 2015, 1–28

RESEARCH ARTICLE

DISCRN: A Distributed Storytelling Framework for Intelligence Analysis

Manu Shukla[a*] and Ray Dos Santos[b] and Feng Chen[c] and Chang-Tien Lu[a]

[a] *Virginia Tech, Fall Church, VA, USA*; [b] *US Army Corps of Engineers - GRL, Alexandria, VA*;[c] *SUNY Albany, Albany NY*

(*v1.0 released April 2015*)

Storytelling connects entities (people, locations, organizations) using their observed relationships to establish meaningful stories among them. Extending that, spatio-temporal storytelling incorporates spatial and graph computations to enhance coherence and meaning. These computations become a bottleneck when performed sequentially as massive number of entities make space and time complexity untenable. This paper presents DISCRN, a distributed framework for performing spatio-temporal storytelling. The framework extracts entities from microblogs and event data, and links those entities to derive stories in a distributed fashion. Performing these operations at scale allows deeper and broader analysis of storylines. This work extends an existing technique based on *ConceptGraph* and *ConceptRank* applying them in a distributed key-value pair paradigm. The novel parallelization techniques speed up the generation and filtering of storylines on massive datasets. Experiments with *Twitter* data and GDELT events show the effectiveness of techniques in DISCRN.

Keywords: Distributed Computing, Spatio-temporal mining, Storytelling, MapReduce

*Corresponding author. Email: mashukla@vt.edu

ISSN: 1365-8816 print/ISSN 1362-3087 online

DOI: 10.1080/1365881YYxxxxxxxx
http://www.informaworld.com

1. Introduction

We have entered the era of events exploding on social media sites such as *Twitter* long before they are picked up by traditional media. With 288 million active monthly users and 500 million tweets a day worldwide in 2015, the activity on *Twitter* is tremendous and growing fast. The computing cost of monitoring rapidly evolving events on *Twitter* and analyzing emerging storylines in space and time makes it imperative to explore parallel computing paradigms.

When a story such as the *Boston Marathon* bombings of April 15, 2013 broke out on *Twitter*; many storylines arose; several people were detained near the blast spots; the residence of a Saudi national was searched; MIT police officer S. Collier was killed; the Tsarnaev brothers were identified as two suspects. All these developments could be observed on *Twitter*, but finding them became impractical under high data volumes. Storytelling, thus, becomes computationally intensive and constrained by slow performance in traditional sequential processing. For example, it would be challenging for an analyst to examine the passing of a new law and the reactions it provokes, such as protests in nearby areas, in a relatively short amount of time. Storytelling as such falls within the field of exploratory analysis where the main focus is discovering new patterns of knowledge or exhaustively analyzing all connections for an entity and then scaling the discovery so it can be performed on a larger dataset rapidly to catch emerging storylines in nascent stage.

In this paper, storytelling is performed according to the technique of Santos *et al.* (2013) which builds a graph of entities (a ConceptGraph) and uses ConceptRank to build the storylines (ConceptRank is a variant of PageRank). One of the most challenging aspects of spatio-temporal storytelling is relationship binding, that is, deciding on how to link entities based on the interactions that they make. This problem arises because there are countless ways of connecting entities: because the are mentioned in the same document, because they are spatially close, because they share similar characteristics, because they were observed talking to each other, etc. For different applications, some of these interactions may be important, while for others, they may be totally irrelevant. From a research perspective, there is no clear method to differentiate them, other than targeting very specific applications, which diminishes the generality of the approach. One way to minimize this problem is to apply distributed processing in order to introspect as many different types of entities and connections as possible, and generate different storylines based on different entities and connections. In this manner, the analyst, and not the algorithm, would be able to decide which entity interactions are important and which are not. Take, for instance, several storylines generated from tweets regarding the Charlie Hebdo attacks in Paris in January 2015.

(1) islamic terrorist $\xrightarrow{mentions}$ isis $\xrightarrow{hacked}$ france $\xrightarrow{investigates}$ possible cyber attack

(2) islamic terrorist $\xrightarrow{mentions}$ isis $\xrightarrow{hacked}$ france $\xrightarrow{defected}$ soldiers

(3) islamic terrorist $\xrightarrow{mentions}$ isis $\xrightarrow{hacked}$ france $\xrightarrow{s'attaque}$ autocensure

In the above three examples, an investigator may be interested in storylines 1 and 2, as they appear possibly related to further cyber and military attacks, while storyline 3 is obtained from tweets that are related to journalism and hence of less pressing concern. In the presence of millions of such storylines from tweets, distributed storytelling allows most of these connections to be presented and reasoned over.

A comprehensive presentation of such interactions is essential not only for completeness

Figure 1.: Large number of storylines generated by DISCRN for evaluation by analyst. Some storylines are more readily actionable than others but analyst evaluates all.

of the facts (e.g., finding lethal activities that may follow terrorist attacks), but also for timeliness (e.g., preventing the acts from occurring). The above examples underscores the need for distributed and parallelized computations, which can enable storytelling with high accuracy and relevance in the presence of massive amounts of data. Figure 1 shows the deeper analysis possible in distributed storytelling.

To perform these tasks at scale creates a number of issues: **Challenge 1) Distributing sequence of steps in storyline creation**. Storytelling involves several steps such as geo-coding locations, extracting entities and relationships from text and binding entities. Performing them at scale requires gluing the pieces together across multiple distributed jobs. **Challenge 2) Scaling graph algorithms utilized in storylines generation**. Calculating ranking and generating storylines using graphs is inherently hard to distribute. With large number of edges and nodes for large datasets it becomes crucial to distribute the process in order to scale. **Challenge 3) Distributed geo-coding and associating locations with entities in storylines**. Geocoding each location from a tweet can result in large number of redundant geocoding requests causing performance bottlenecks. Geocoding and combining geocoded location with storylines generation is key to scaling. **Challenge 4) Deeper analysis by allowing analyst to focus on sections of large number of storylines**. It is interesting to find storylines that are relevant beyond pure ranking and push their relevance higher to allow analyst to search through them. Researching these techniques is crucial to take full advantage of large number of storylines generated by distributed methods.

Performing storytelling at scale can be achieved through several means. Previous research in distributed spatio-temporal mining have used techniques such as MPI and CUDA on gpu clusters as described in Qin *et al.* (2014) and Osterman *et al.* (2014). In HPC the most commonly available tool for distribution is key-value pair paradigm. Various open source frameworks are readily available to distribute mining horizontally based on key-value pairs. CUDA and other tools require specialized hardware and programming techniques for distribution that are not widely applicable for other distribution problems. The work in this paper models the entire storytelling process in key-value pairs and implements it in popular and widely available MapReduce paradigm. A distributed framework DISCRN is presented as a means to generate storylines using spatio-temporal techniques on short, ill-formed text of *Twitter* data . Its efficacy is proved on structured

pre-formatted event data in GDELT. As with *ConceptGraph* based approach in Santos *et al.* (2013), key to obtaining coherent stories is to identify regions of spatial propagation where related entities cluster. However there is tremendous value in creating exhaustive list of storylines and allowing analysts to sift through them as needed to discover new emerging ones. We use big data tools on the cloud at our disposal that provide key-value pair based distribution to perform these tasks at scale.

The key contributions of the paper can be summarized as follows:

(1) **Distributed computing techniques to create and analyze storylines at scale**: Extracting entities from tweets, computing ConceptRank and computing storylines in parallel allows the stories to be created at a large scale. Novel distribution algorithms to perform the computations are presented.
(2) **Novel ConceptGraph traversal to generate storylines**: Distributed storyline generation requires traversing *ConceptGraph* which is a graph of entities. A novel traversal technique *ConceptSearch* is presented that performs distributed forward and bi-direction graph search to generate storylines from directed *ConceptGraph*.
(3) **Deeper analysis through Spatial filtering with large number of storylines evaluated**: The approach with iterative *ConceptSearch* produces storylines for multiple starting points for evaluation by analyst with little incremental computation cost for each additional starting point. Filtering on entity network distances allows analysts to focus on smaller set of storylines.
(4) **Validation on multiple large scale datasets**: Experiments are performed on *Twitter* and GDELT datasets. The results validate the distribution techniques and the ability to extract deeper more meaningful results. These insights are not obtainable from sequential storytelling.

The rest of the paper is organized as follows. Section 2 elaborates on existing work, highlighting differences and specifically the need to scale to large data sets. In Section 3, overview of key-value pair based distribution and its use in DISCRN architecture and flow are provided. Section 4 gives detail of key-value pair based algorithms implemented in DISCRN including details of the *ConceptSearch* algorithm and distributed entity-network based filtering based on *inverted value join* technique. Experiments are presented in Section 5 and conclusions provided in Section 6.

2. Related Work

This section reviews works similar to concepts proposed in this paper that span many areas of expertise such as storytelling, traversing graphs and analyzing sequences of entities spatially in parallel. Section 2.1 reviews prior work in field of storytelling and Section 2.2 presents work in distributed spatio-temporal mining.

2.1. *Storytelling*

Storytelling is the process of connecting entities through their characteristics, actions, and events in Turner (1994). *Information retrieval* and web research have studied this problem, i.e., modeling storylines from search results, and linking documents into stories in Kumar *et al.* (2008), Hossain *et al.* (2011) and Hossain *et al.* (2012b) (the terms *stories* and *storylines* are used interchangeably). Traditional *storytelling* attempts to link

disparate entities that are known ahead of time, such as the connections between two individuals. Beyond traditional text analysis, spatio-temporal entity analysis has been explored in Santos *et al.* (2012), which can fill some of the gaps left by traditional approaches. ConceptRank based storytelling is explored in Santos *et al.* (2013). Forecasting events based on spatio-temporal storytelling is described in Santos *et al.* (2014).

Linking entities across documents has been done successfully using distributed inference technique in Singh *et al.* (2011). Named entity extraction techniques have been exhaustively surveyed in Finin *et al.* (2010). Leveraging knowledge bases to extract entities has been explored in Michelson and Macskassy (2010). Link analysis algorithms often rely on graphs as a modeling abstraction, such as the evolution of entities in space and time (Mondo *et al.* (2013), Chan *et al.* (2009)) and the identification of patterns (George *et al.* (2009), Chan *et al.* (2008)). The spatio-temporal aspects observe how entities propagate. Twitter data analysis for storytelling has been explored. GDELT is an event database that is growing rapidly. GDELT data being structured does not need preprocessing steps and distances are feasible as the geo-coordinates of actors and actions in each events are explicitly provided in Leetaru and Schrodt (2013). GDELT data has been used for several works in spatio-temporal events analysis.

2.2. *Distributed graph and spatio-temporal analysis*

Connecting the terms to form storylines has been an established technique for a while with several notable techniques proposed in terms of multiple entities in Hossain *et al.* (2012a) or fewer in Shahaf and Guestrin (2012). Associating users through tweets has also been studied in Weitzel *et al.* (2012). There are several distributed graph traversal algorithms. Some have been performed on GPUs in Merrill *et al.* (2012). Partitioning graphs also aids distribution and traversal in Bulu and Madduri (2012). A very important aspect of our approach is an algorithm to perform ConceptRank computations over ConceptGraphs in parallel. Distributed computation paradigms such as MapReduce do not easily lend themselves to parallel *ConceptRank* computation. However recent works have explored parallel PageRank computations in great details with tremendous success in Lin and Schatz (2010). Other works that have explored MapReduce and key value pair based random walk distribution have taken Monte Carlo based approach in Sarma *et al.* (2012)

Distributed storytelling proves to be highly effective on massively sized datasets such as Twitter which are destined to get bigger, along with other sources of social media data. Key to it is scalable joins. Ways to distribute data joins across multiple sources using key value pairs are explored in Afrati and Ullman (2010). However these joins are not sufficient for storytelling purposes. As far as we know there are no distributed storytelling techniques discussed in research so far.

3. Distributed Storylines generation platform

In this section, various techniques used in building storylines and their parallelization are described. Building storylines from social media data where news events first emerge requires scaling computations to large amounts of data. Here, key-value pair based distribution techniques are used to generate storylines at scale in DISCRN. DISCRN architectural details are provided in Section 3.1 including a brief overview of Hadoop MapReduce. Section 3.2 describes the parallelization flow in DISCRN.

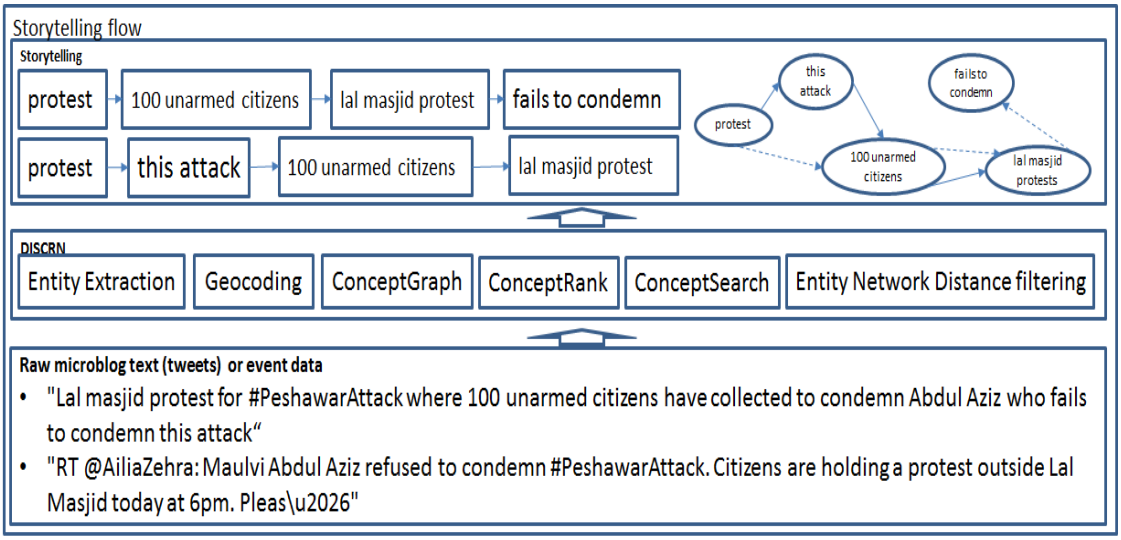

Figure 2.: Storylines building process from raw microblog data using DISCRN with input tweets and output storylines.

3.1. *DISCRN Architecture*

This section describes the architecture of DISCRN. It is based on key-value pair parallelization of steps on MapReduce platform. The generation of storylines from raw microblog or event data is described in Figure 2. Briefly storytelling involves 5 key tasks, extraction of entities, geo-coding locations, building a graph of entities and relationships, ranking, traversing graph to generate storylines, and storyline filtering based on entity distances.

Apache and Hadoop (2014) is an open source framework which facilitates distributed computations on large clusters. A master node orchestrates data storage and computation distribution over multiple slave nodes. Files are uploaded into distributed file storage called HDFS, split into 64MB blocks and then processed. Master node keeps track of all the blocks of a file and where they are stored. MapReduce in Dean and Ghemawat (2008) allows master node to break down computation tasks into *mappers* and *reducers* distributed over slave nodes. They work on file blocks on slave nodes exploiting co-location of computation with data. Mappers read in input data as key value pairs $<k_1, v_1>$and emit intermediate key value pairs $<k_2, v_2>$. Reducers receive the intermediate key value pairs grouped by k_2 and processes them to generate the final set of key value pairs $<k_3, v_3>$. The keys k_2 are sorted and custom partitioners can determine which reducer handles a particular key-value pair.

Key-value pair based parallelization allows data records to be split into key-value pair elements that can be then operated upon independently in parallel with other records on multiple nodes of a cluster. The challenge then becomes to associate all sub-elements needed to perform an operation to generate a result element in the value of key-value pair and emit it with a key that would uniquely identify that operation. The operation can then generate a single result value which in combination with other result values will form the result set needed from the input data. If a result set r_i from an input set e_i is needed then the key-value pair based paradigm requires emitting each input record e_i as key value pair <k,v>. All the elements needed to generate a result value r_i from e_i will be in the emitted value v_i with the key k_i being unique so as to distribute pairs across nodes without collisions. In most operations it is not feasible to build the key-value pair

o

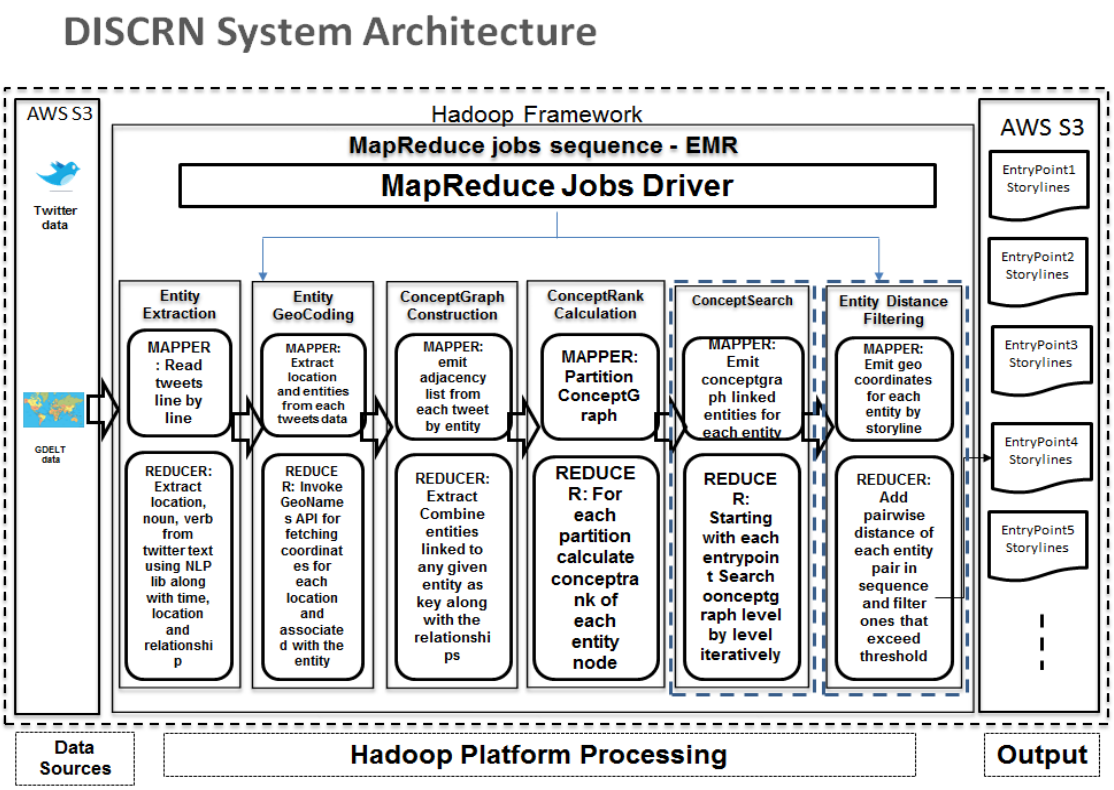

Figure 3.: Platform for distributed computation of storylines. Boxes are the key MapReduce jobs needed to calculate the storyline.

to convert an input value to a result value in a single step. It is then required to perform a series of the key-value pair emissions and keep sub-processing or aggregating the elements in the value so as to bring it to the state where it can perform the final operation to emit the result value. In simple cases, that is similar to a multi-key join where multiple inputs combine to create a single value in order to perform an operation. In most cases however complex computations need to be performed to bring the intermediate values to a state that can generate a result value.

The MapReduce jobs for algorithms described in Section 4 are implemented in DISCRN. Architecture of DISCRN is shown in Figure 3. It shows the key MapReduce jobs that are run in sequence to produce the outputs for storylines. A driver calls each job in a sequence and decides which modules need to be executed. Data is written to disk at the end of each MapReduce job and then read again in the subsequent one. Steps with dotted line boxes indicate multiple jobs. However the number of MapReduce jobs is always bounded and small enough to not have disk I/O as a major issue as each of subset of inputs are being read on separate cluster nodes. A sequence of jobs implement the key-value pair based transforms described in following section. The spatio-temporal techniques including extracting locations from tweets along with entities and relationships which constitute the first set of jobs is performed. The locations are then used to fetch the geo-coordinates of the locations that get associated with the entities and used to find spatial distance between entities. Subsequently a ConceptGraph is built with the entities as nodes and predominant relationship between entities as edges. The ConceptRank of all the entities is calculated and associated with each entity of storyline. The last two

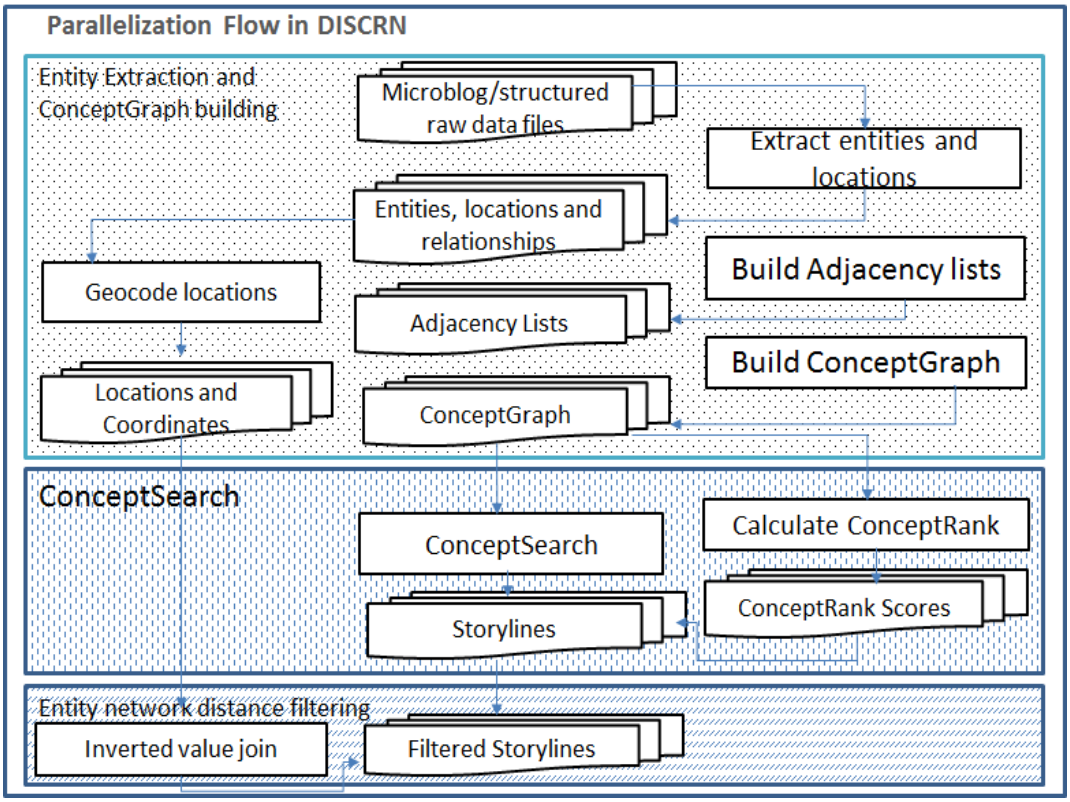

Figure 4.: Parallelization Flow to build storylines. Key steps that are parallelized and data from each stage joined and used in subsequent tasks.

sets of jobs perform iterative ConceptSearch to generate the storylines and then filter them based on sum of pairwise entity distance. The final output is a set of storylines filtered by distance. Distance based filtering can be turned off as well if not needed by analyst.

The number of MapReduce jobs stays constant for storylines of certain length in DISCRN. The data written to disk is generally a subset of raw tweets and GDELT data that is initially used in processing. The larger the number of jobs, the more times disk is hit. This however is a less significant expense as most data is generated for specific reasons such as geo-coordinates of entities that is later joined with other datasets such as entities in storylines to perform entity network distance based filtering or combining storylines with the ConceptRank of entities. They are later used in sorting and filtering of storylines.

3.2. *Parallelization Flow in DISCRN*

This section describes the flow of data through various steps starting from raw microblog data to completed storylines. A parallelization flow of the entire storyline building process is shown in Figure 4. The flow is a composite of 3 parts, the pre-processing, the ConceptRank and ConceptSearch components and the joins based entity network distance filtering. This flow allows for key-value pair based parallelization of each step, and then combining them together. Each step of the flow is one or more mapreduce jobs. In Figure 3 the first set of 3 MapReduce jobs are for preprocessing. They include en-

tity extraction, geocoding and ConceptGraph building. The ConceptRank calculation and ConceptSearch belong to the ConceptRank and ConceptSearch block. The Entity Distance Filtering jobs map to the Entity Network Distance Filtering component.

The **Entity Extraction and ConceptGraph building** component describes how distributed entity extraction and geocoding extracts entities in parallel on cluster nodes to create adjacency lists and geocode locations to associate each entity with coordinates. Geocoding parallelizes by fetching coordinates of locations from *geonames* services in GeoNames *et al.* (2015) in parallel once and then joining with them subsequently in operations where coordinates of entities are needed such as entity network distance computations. The adjacency lists from each tweet are consolidated into a ConceptGraph that has each entity as node and entities it has outgoing nodes to as connected nodes. This graph is then used to calculate ConceptRank using modified Shimmy pattern MapReduce jobs in Lin and Schatz (2010).

The **ConceptSearch** module computes ConceptRank in parallel from ConceptGraph and then iterates over ConceptGraph with starting entities k times to build storylines of length k entities. ConceptSearch is called iteratively such that at the end of each iteration the length of candidate storylines is incremented by 1 and after final iteration storylines of length k is output. ConceptSearch can traverse ConceptGraph both in forward direction following outgoing links or in bi-directional pattern by following both outgoing and incoming links.

The **Entity network distance filtering** component uses *inverted value join* to combine locations and entities of storylines together to filter out storylines with entity pair distance sum exceeding a threshold. Threshold is provided by user as parameter. The joins are performed such that there is no need to build large memory structures at any stage to ensure full scalability. Entity network distance based filtering can be skipped and storylines will still be generated without the filtering.

4. Storytelling distribution in DISCRN

This section presents the key algorithms in Storytelling and their distributed implementations in MapReduce. Section 4.1 describes preliminary processes of storytelling such as entity extraction, location geocoding and combining adjacency lists from tweets into ConceptGraph. Section 4.2 describes ConceptSearch in MapReduce and Section 4.3 gives entity network distance based filtering using entity coordinates and storylines in MapReduce.

4.1. *Entity extraction, geocoding and building ConceptGraph in MapReduce*

Algorithm **1** describes the 3 mapreduce jobs used to perform entity extraction, geocoding of locations and building ConceptGraph from raw microblogs input. The first step in building storylines is extracting entities from micoblogs. Modeling that process as key-value pairs is in MapReduce1. Parsing text and extracting entities is shown in line 3. Here e_i:r_i are the entity and relationships extracted from text of the microblog snippet $content_i$, as noun and preceding verb, loc_i is the location of post and t_i the time. This step builds adjacency lists from extracted entities. Adjacency lists consists of entities and relationships extracted from a tweet as nouns and verbs. The first entity is the starting node in the adjacency list and all the subsequent entities and relationships are the nodes

with links coming in from the starting entity. The second step is geocoding each location from microblog entry. That process is modeled as key-value pairs in MapReduce2 as shown in line 10. The third step is consolidating adjacency lists into ConceptGraph nodes. Modeling that process as key-value pairs requires aggregating values of a key representing an entity in MapReduce3 in algorithm. Here e_j is the starting entity in a tweet and e_c:r_c;e_d:r_d... are the following entities and relationships extracted from it. e_j from multiple tweets and its corresponding following entities and relationships are combined together in reducer to generate all the nodes for e_j that have incoming links to them as shown in line 15. The most frequent relationship from starting entity to following entities in case of collision are kept. The combined result is a ConceptGraph where each node e_i is connected to a set of edges and verticies (G,V) as ($\{e_i\}$,$\{r_i\}$).

Algorithm 1 Extract entities, geocode locations and build ConceptGraph

1: MapReduce1:
2: Mapper1:
Input:$<k_1, v_1> \rightarrow <offset_i, content_i>$
3: Parse $content_i$ and extract $\{e_i : r_i\}$ // using nlp parser and classifier
Output:$<k'_1, v'_1> \rightarrow <tweetid_i, e_1{:}r_2; e_2{:}r_2; ...; loc_i; t_i>$
4: Reducer1: {I}dentity reducer
Input:$<k'_1, v'_1> \rightarrow <tweetid_i, e_1{:}r_2; e_2{:}r_2; ...; loc_i; t_i>$
5: parse tweet //simply output the tweetid and the entities, relationships, locations, timestamp,...
Output:$<k'_1, v'_1> \rightarrow <tweetid_i, e_1{:}r_2; e_2{:}r_2; ...; loc_i; t_i>$
6: MapReduce2:
7: Mapper2:
Input:$<k_1, v_1> \rightarrow <tweetid_i, e_1{:}r_1; e_2{:}r_2; ...; loc_i; t_i>$
8: emit each entity e_i in tweet by location loc_i // assume each entity in tweet has same location
Output:$<k'_1, v'_1> \rightarrow <loc_i, e_1{:}e_2{:}...{:}e_n>$
9: Reducer2:
Input:$<k'_1, v'_1> \rightarrow <loc_i, e_1{:}e_2{:}...{:}e_n{:}...{:}e_s>$
10: Geocode loc_i to $coord_i$ using geonames api // api is called for each location once saving compute and money
Output:$<k'_1, v'_1> \rightarrow <e_1, loc_i{:}coord_i>$
$<e_2, loc_i{:}coord_i>$
....
$<e_s, loc_i{:}coord_i>$
11: MapReduce3:
12: Mapper3:
Input:$<k_1, v_1> \rightarrow <tweetid_i, e_a{:}r_a; e_b{:}r_b; ...>$
$<e_j, e_c{:}r_c; e_d{:}r_d ...>$
13: Separate and emit each entity pair $\{e_i\ e_a : r_a\}$ from adj list
Output:$<k'_1, v'_1> \rightarrow <e_i, e_a{:}r_a>$
$<e_i, e_b{:}r_b>$
$<e_i, e_c{:}r_c>$
...
14: Reducer3:
Input:$<k_1, v_1> \rightarrow <e_i, e_a{:}r_a>$
15: Combine adjacency lists entities e_i,$\{e_a{:}r_a ...\}$ into *ConceptGraph* // each entity can have multiple values that need to be iterated over and combined
Output:$<k'_1, v'_1> \rightarrow <e_i, e_a{:}r_a; e_b{:}r_b; e_c{:}r_c ...>$

Fourth step is passing ConceptGraph nodes to ConceptRank calculation algorithm. This is modified Shimmy pattern based algorithm that partitions graph and iteratively calculates ConceptRank of each node. The algorithm has 3 steps, first ingests the graph on which to calculate ConceptRank, second divides graph into partitions, third iterates over partitions calculating ConceptRank of each node. The ConceptGraph is then passed into the *ConceptSearch* traversal algorithm with the corresponding nodes ConceptRank values associated with the entity nodes through a join.

4.2. *ConceptSearch in MapReduce*

This section describes the ConceptSearch algorithm implemented in MapReduce using key-value pair based distribution. The first entity in each microblog text or event actor list in GDELT is considered a starting entity and all the following entities are supposed

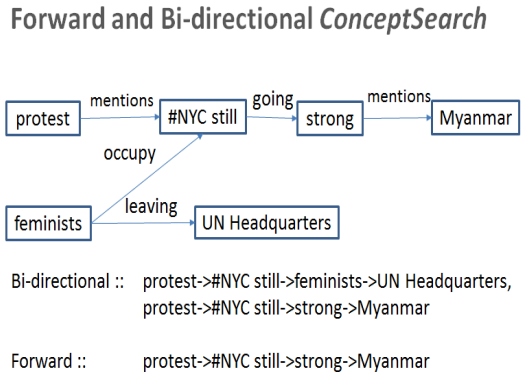

Figure 5.: Bi-directional and forward only ConceptSearch. In forward traversal, each entity is a starting entity in a tweet and entities are connected in strict sequential mode representing outgoing direction of arrows. In bi-directional traversal, direction does not have to follow outgoing arrows, 'feminists' is the first entity in a tweet and has outgoing edge to '#NYC still' and can still be part of a storyline.

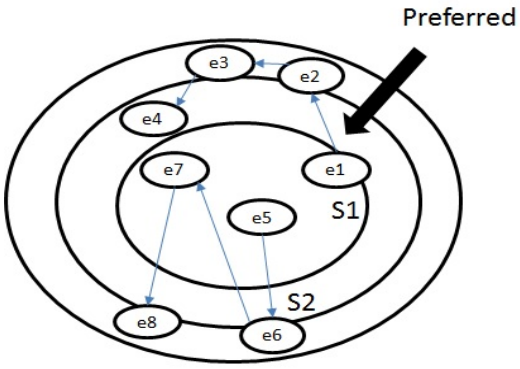

Figure 6.: The circles show the geographical regions over which entities are scattered. Two storylines are depicted: S1 links "'e1-e1-e3-e4"' while S2 is "e5-e6-e7-e8". Looking at each pair of connected entities in the two storylines, it can be seen that storyline S1 has a total pairwise distance between entities less than the storyline S2. That makes S1's geographic footprint smaller, which in many intelligence analysis applications is importance to analysts.

to have incoming links from starting entity. ConceptSearch has two variations.

(1) Forward Traversal: Traversing the ConceptGraph by only following the outgoing edges forces traversal to be only from the starting node of an adjacency list on to the secondary entities.
(2) Forward and Backward bi-directional Traversal: Traversing the ConceptGraph by following the outgoing and incoming edges allowing traversal to be from any node of an adjacency list on to the starting or secondary entities.

Figure 5 illustrates the two different types of storylines produced by the bi-directional and forward only ConceptSearch. The figure shows two example storylines with starting entity 'protest', the first storyline *protest* $\rightarrow$ *#NYC still*$\rightarrow$*strong*$\rightarrow$*Myanmar* gives a list of connected entities that appeared in tweets with starting entity as protest, followed by '#NYC still'. In another tweet with starting entity as '#NYC still' and following entity as 'strong', and in a third tweet starting entity as 'strong' followed by 'Myanmar'. In the other storyline however 'feminist' can be any entity in the storyline provided it also had term '#NYC still' in it, and again 'UN Headquarter' can also be anywhere in a tweet text as an entity along with 'feminists' in it.

For large datasets, ConceptGraph can not be held in memory on a single node and needs to be created and traversed on a cluster over large number of nodes. Earlier in this section the preprocessing for k entity storylines construction algorithm using the key-value pair based distribution is described. In distributed forward search described in Algorithm **2** the storylines can be generated by traversing the ConceptGraph by only following outgoing links. The starting entities are read into distributed cache and the ConceptGraph which is a set of nodes and its corresponding outgoing edges to connected

Algorithm 2 Forward ConceptSearch

```
1: MapReduce4
2: Mapper4
     Input:<k1, v1>→ <ei,ej;ek;el...>// each graph node and its connected nodes
      distributed cache: → se1, se2, se3..., sen
3: for all sei in se1, se2, se3..., sen do
4:     for each starting entity
5:     if sej == ei then
6:         for each starting entity // match starting entity with key
7:         for all ek in k1 or v1 do
8:             match entity with any following entity for combinations
                 Output:<k'1, v'1>→ <sej,{ea}>{ea is the list of all entities other than sej}
9:         end for
10:    end if
11: end for{initial extraction}
12: Reducer4 {combine elements from multiple nodes and emit a consolidated list per node}
     Input:<k'1, v'1>→ <sej,{ea}>
13: combine ConceptGraph entities ei,ej;ek;el... and connected entities with starting entities sei
     Output: <k'1, v'1>→ <sej,{ek}>{finish init loop}
14: for all i=1;i<k;i++ do
15:    //iterate for storylines of length k
16:    MapReduce5:
17:    Mapper5:
        Input:<k1, v1>→ <ei,ej;ek;el...>
18:    prefix value with 'entities::'
19:    append '::' to key
        Output: <k'1, v'1>→ <ei ::,entities::{ei; ej, ...}>
          <ej ::,entities::{ei; ej, ...}>
          ....
20:    Mapper5':
21:    prefix value with entities 'storylines'
        Input: <k2, v2>→ <e'i,sj;sk;sl...>
       {sj is a sub-story which is a combination of entities of format ej : ek...}
        Output: <k'1, v'1>→ <e'i,storylines::{si; sj; ...}>
          <e'j,storylines::{si; sj; ...}>
          ....
       {second loop} {The second job is called iteratively to aggregate incrementally storyline entities in the key.}
22:    Partitioner5:
        Input:<k1, v1>,numReduceTasks → <ei[::],entities::|substories::ei; ej...|si; sj...>
23:    extract entity from key
        Output:hashCode(ei)%numReduceTasks
24:    Reducer5:
        Input:<k1, v1>→ <ei[::],<ei[::],entities::|substories::ei; ej...|si; sj...>
       {second loop}
25:    for all i=1..n do
26:        if ei == k1 then
27:            second loop // match starting entity
28:            for all ek in k1 or v1 do
29:                emit extended storylines
                     Output:<k'1, v'1>→ <sj:ei,{ek}>// pull in entities in the value list
30:            end for
31:        end if
32:    end for
33: end for
```

nodes is emitted from mapper and checked for the starting entity being the source node as shown in line 6. If so, the storylines with each of the connected nodes are created. The next set of iterations then starts until we have from the potential storylines list and progressively following to the next level of corresponding entities for the nodes storylines to a specified level. In MapReduce5 the partitioner in line 23 exploits the ordering of keys by MapReduce to prevent the buildup of all possible connected entities and substorylines in a reducer process. This traversal restricts storytelling creation to only storylines that have first entity in a microblog text snippet as the starting entity selected by user. This may be too restrictive for some domains but desirable in others.

Algorithm **3** traverses ConceptGraph bi-directionally to produce storylines as shown in Figure 5. In bi-directional ConceptSearch the traversal of conceptgrah at each iteration can begin from starting node or connected node and the next connected node can be either a starting node or any node in its list of connected nodes. This invariably results in much larger number of storylines. The *ConceptSearch* algorithm iteratively searches

Algorithm 3 Bi-directional ConceptSearch

```
1: MapReduce4
2: Mapper4
     Input: <k1, v1>→ <ei,ej;ek;el...>// graph node and its connected nodes
     cache → se1, se2, se3..., sen
3: for all sei in se1, se2, se3..., sen do
4:     for each starting entity
5:     if sej == ei OR sej in ej,ek,el then
6:         for each starting entity // match starting entity with any key or value entity of ConceptGraph node and
           connecting nodes
7:         for all ek in k1 or v1 do
8:             // match entity with any entity for combinations
                 Output: <k'1, v'1>→ <sej,{ea}>{ea is the list of all entities other than sej}
9:         end for
10:    end if
11: end for{initial extraction}
12: Reducer4 {combine elements from multiple nodes and emit a consolidated list per node}
     Input: <k'1, v'1>→ <sej,{ea}>
13: combine ConceptGraph entities ei,ej;ek;el... and connected entities with starting entities sei
     Output:<k'1, v'1>→ <sej,{ek}>{finish init loop}
14: for all i=1;i<k;i++ do
15:    //iterate for storylines of length k
16:    MapReduce5:
17:    Mapper5:
         Input:<k1, v1>→ <ei,ej;ek;el...>
18:    prefix 'entities' to value
         Output: <k'1, v'1>→ <ei ::,entities::{ei; ej, ...}>
           <ej ::,entities::{ei; ej, ...}>      ...
19:    Mapper5':
         Input: <k2, v2>→ <e'i,sj;sk;sl...>{sj is a sub-story which is a combination of entities of format ej : ek...}
20:    Append '::' to entity in key and prefix value with 'substories::'
         Output:<k'1, v'1>→ <e'i,substories::{si; sj; ...}>
           <e'j,substories::{si; sj; ...}>
           ...
       {second loop} {The second job is called iteratively to aggregate incrementally storyline entities in the key.}
21:    Partitioner5:
         Input:<k1, v1>,numReduceTasks → <ei[::],entities::ei; ej...|substories::|si; sj...>
22:    extract entity from key
         Output:hashCode(ei)%numReduceTasks
23:    Reducer5:
         Input: <k1, v1>→ <ei[::],entities::ei; ej...|substories::|si; sj...>
       {second loop} {tertiary loop}
24:    for all i=1..n do
25:        if ei in k1 or in v1 then
26:            second loop // match starting or any subsequent entity
27:            for all ek in k1 or v1 do
28:                emit extended storylines
                     Output:<k'1, v'1>→ <sj:ei,{ek}>// pull in entities in the value list
29:            end for
30:        end if
31:    end for
32: end for
```

exhaustively for all entities and links by iterating over ConceptGraph nodes. The algorithm starts by reading set of starting points into cache and then checking each starting node and connected nodes to see if any of the nodes match a starting entity as shown in line 6. If so it creates a list of partial storylines that in next iteration are built to next level by checking all the subsequent level nodes and the appearance of any of the last nodes in the starting or connected list. These operations can be performed iteratively by calling the same key-value pair based traversal for each step of storylines extension.

A key advantage of distributed ConceptSearch is that each iteration traverses all nodes and starting entities in parallel. This facilitates traversing ConceptGraph and creating storylines for multiple starting points. As ConceptGraph is created first and during traversal multiple starting nodes or last nodes of partial storylines can be considered for traversal to next level the incremental cost of additional starting points during each level of storylines building during a traversal iteration is small.

Table 1.: Twitter Storylines where one meets the user provided entity network distance threshold and one exceeds it.

	Outside 100km threshold	**Within 100km threshold**
1	protest 41.4822:-81.6697 → america 41.4808:-81.8003 → the realm 41.3996:-81.6457→ la chine 51.89842:4.50328	protest 41.4822:-81.6697 → america 41.4808:-81.8003 → selma protesters 41.3996:-81.6457 → the chants 41.4172:-81.6722

4.3. *Entity network distance based filtering in MapReduce*

This section describes a technique to focus storylines to ones that have the sum of pairwise distance between entities within a certain threshold. This provides for the analyst to focus on storylines with geographical footprint within a threshold. Figure 6 shows the filtering of storylines by entity network distance threshold. The operation is key in limiting the number of storylines an analyst has to explore and focusing on entities in a particular geographic area. A spatial entity is one for which location can be obtained, while temporal entities are associated with a timestamp. When neither location nor time is available, the entity is denoted as purely textual. The claim that space and time are important to storytelling leads to investigating the influence of spatio-temporal entities on the storylines by calculating distances between entities as their spatial distances assuming temporal ordering using Time Matrix in Santos *et al.* (2013). This spatial distance between entity pairs is used to filter storylines. The spatio-temporal filtering allows analyst to focus on a certain segment of storylines. An example of a storyline that meets the 100km threshold requirement and another that exceeds the threshold is shown in Table 1. The storyline that exceeds the threshold has one of its entities 'la chine' in Nigeria while the remaining entities are in Ohio, USA. Entity network distance filtering requires identifying and applying the best coordinates for an entity in a storyline. It utilizes *inverted value join* where the elements of values of key storyline is emitted and then enriched and combined again into the key. This is a classic example of multi-stage value preparation in order to consolidate all values needed to perform entity network distance geo-coordinate calculation on a single key value pair as one operation. The details of key-value pair based operations are shown in Algorithm **4**.

Mapper6 and Reducer6 in Algorithm **4** start with input as set of key-value pairs representing a storyline and its corresponding entities and relationships. It then splits a storyline into a set of entity and storyline key-value pairs. Mapper7 combines an entity and its coordinate location as key-value pairs that are then combined with the entity and all the storylines it appears in to generate a list of storylines and the coordinates of each entity. Reducer7 then creates a key-value pair for each storyline in an entity value as key along with the entity and coordinate location as value. Multiple key-value pairs with key as storyline are then combined together to give the entire list of entities along with their coordinate locations as value for a storyline key S_i. The final transform in Reducer8 generates key-value pair with storyline S_i as key only if its sum of entity pair distances $entitydist_i$ is below a threshold. The sequence in MapReduce7 and MapReduce8 in figure illustrates the *inverted value join* operations in MapReduce that is utilized to perform entity network distance based filtering. In MapReduce implementation of storyline network distance computation several joins are needed to perform the network distance based storyline culling. With each entity details large number of keys are emitted during the generation of entities within storylines. Those keys are essential to fully parallelize the pairwise network distance computation, and are required to land all the requisite entities and their locations as values for a storyline key in a reducer after multiple mapreduce jobs. In order to emit each set of entities for storylines for a

Algorithm 4 Inverted value join

1: MapReduce6:
2: Mapper6:
Input:$<k_1, v_1> \rightarrow <S_i, e_1{:}r_1; e_2{:}r_2; ....e_n{:}r_n>$
3: emit each entity e_i with its corresponding storyline S_i
Output:$<k_1', v_1'> \rightarrow <e_1, S_i>$
$<e_2, S_i>$// emit each entity and storyline its in separately
....
4: Reducer6: // remit all storylines for an entity
Input:$<k_1, v_1> \rightarrow <e_i, S_j>$
5: emit the storylines for each entity e_i with suffix
Output:$<k_1', v_1'> \rightarrow <e_i{::}, S_i>$
6: MapReduce7:
7: Mapper7: // join with pre-fetched coordinates as input
Input1:$<k_1, v_1> \rightarrow <e_i, coord_i>$
8: emit the coordinates for each entity e_i with suffix
Output:$<k_1', v_1'> \rightarrow <e_i$,Location::$coord_i>$
9: Mapper7': // emit each entity and its corresponding storylines as is from previous job
Input2:$<k_2, v_2> \rightarrow <e_i{::}, S_1>$
$<e_i{::};S_2;...>$//emit the entities as is in with suffix
$<e_i{::};S_n;>$
10: emit the storylines for each entity e_i with suffix
Output:$<k_1', v_1'> \rightarrow <e_i{::}, S_1>$
$<e_i{::}, S_2>$// output each entity with suffix with storyline
....
$<e_i, S_n>$
11: Partitioner7:
Input: $<k_1, v_1>$,numReduceTasks $\rightarrow <e_i[::], coord_i|S_i>$
12: direct each entities location and storylines to same reducer // take advantage of key sorting in MapReduce
Output:hashCode(e_i)%numReduceTasks // ensures that coordinates are ahead of entities and storylines in sorted key list and each entity key value pairs sent to same reducer
13: Reducer7:
Input:$<k_1, v_1> \rightarrow <e_i, S_1; S_2; ...S_n>$
$<k_1, v_1> \rightarrow <e_i{::}, coord_i>$
14: combine location with storylines of keys as they come in order location first with storylines following
Output:$<k_1', v_1'> \rightarrow <S_1, e_i; coord_i>$
$<S_2, e_i; coord_i>$//emit each storyline with its constituent entity and coordinate
....
$<S_n, e_i; coord_i>$
15: MapReduce8:
16: Mapper8:
Input:$<k_1, v_1> \rightarrow <S_i, e_j; coord_j>$
17: emit each storyline with entity and location
Output:$<k_1', v_1'> \rightarrow <S_i, e_j{:}coord_j>$
18: Reducer8:
Input:$<k_1, v_1> \rightarrow <S_i, e_j; coord_j{:}e_k; coord_k{:}e_l; coord_l ...>$
19: Combine entities and corresponding location of a storyline
20: Calculate distance //these are coordinate distances not euclidean and
21: **if** dist($e_j : e_k + e_k : e_l + ...$) <threshold **then**
22: compare with threshold //threshold is a parameter passed in provided by user
Output $<k_1', v_1'> \rightarrow <S_i, e_j{:}coord_j; e_k; coord_k ...>$
23: emit only storylines within the threshold
24: **end if**

large dataset of millions of entities and storylines, the number of pairs of entities and storylines will be enormous, which can be a significant performance bottleneck. However since entities are often repeated the *inverted value join* inherently limits the number of keys as entities.

To alleviate issue of having a key with too many values when entities are emitted with all storylines they are a part of, it is crucial to utilize the natural sorting of keys provided in MapReduce. Constructs available in MapReduce paradigm allow us to exploit natural ranking of elements and order within the entities and storylines data to limit our key-value pair emissions and reduce the number of jobs by eliminating ones purely used for sorting. Since the keys are sorted on text fields in default ascending text order, it is useful to leverage that order to give keys with storylines a lower sort order and ones with location a higher sort order so that they can be re-emitted as values for keys as storylines in order to join the values as entities along with their *ConceptRank* and locations. A custom partitioner ensures the keys for an entity go to a particular reducer e.g. <e1, Location::lat;lon>will appear earlier in sorted key list than $<e_1 ::, S_1; ...>$...

which can then be exploited to associate location of e_1 in emission in Reducer6 in line 5. This prevents having to store the storylines in memory until the location value is received potentially causing memory errors. Section 5 shows empirically that the network path based storyline pair culling gives good results on limiting the number of storyline pairs calculated and improving the quality and reducing the quantity of storylines to be reviewed by an analyst.

4.4. *Complexity*

In sequential processing the computation complexity of entity extraction for Twitter data is O(T) where T is the number of tweets, performance of storylines construction is O(N) with order of ConceptRank computations O(N) where N is network size. The complexity of building storylines based on ConceptRank is O(N) or number of nodes in ConceptGraph. In distributed environment, the order of extracting entities is O(T/k) where k is the number of distributed processes, order of building storylines by distance computations is O(N/k), to perform distributed ConceptRank is O(logN/ϵ) where (1-ϵ) is the dampening factor and that of building storylines based on distributed iterative storyline generation is O(N/k). These numbers clearly show the scalability of distribution as the processing can be rapidly scaled by adding nodes with each node running multiple mappers and reducers. For forward versus bi-directional ConceptSearch computation complexity is the same even though vastly different number of storylines are generated in bi-directional ConceptSearch. For multiple starting points the set of starting entities is greater than 1 yet the order of traversal is still O(N/k) as each new starting point does not increase number of passes made through ConceptGraph data.

The space complexity of the distribution techniques is O(E) where E is the maximum number of entities to which a ConceptGraph node has outgoing connections. This is vast improvement over sequential implementation's space complexity which is O(N). The constrained size of data structures on each process also avoids memory issues. MapReduce allows iterating all the connected nodes of a ConceptGraph node without storing them in memory which suffices for operations such as entity network distance filtering. With high memory usage in sequential processing, even if nodes in structures for ConceptGraph include reading in from disk as needed the paging rate increases deteriorating performance further. For multiple starting points memory usage stays constrained to O(E). In some algorithms there is need to combine all storylines that contain an entity such as in *inverted value join* which is of order O(N) which can cause memory errors. Keep space complexity in a process to E which is of lower order than N hence is crucial along with iterating over the connected nodes instead of storing them in a process memory. With the astute use of partitioners the memory footprint in each reducer for *inverted value join* is kept to order O(E) for storylines filtering. The memory footprint of ConceptSearch iterations is also kept as O(E). This allows for node level operations on ConceptGraph be performed in single reducer in ConceptSearch.

The marked improvement in space and time complexity and capping the space complexity in distributed operations even for large datasets that allows the distributed technique to sale to massive datasets.

5. Empirical Evaluation

Experiments for building storylines from Twitter and GDELT data are performed including forward and bi-directional ConceptSearch and entity distance based filtering with pre-determined thresholds in DISCRN. This section presents the experimental validation of scalability and efficiency of storytelling techniques described in previous sections. Our experiments focused on showing the performance enhancements brought about by parallelization of various stages of storytelling. Section 5.1 gives high level overview of experiment design and computing environment used. Section 5.2 provides results of experiments with Twitter data. In Section 5.3 efficiency of distributed *ConceptSearch* and in Section 5.4 entity network distance based filtering of storylines is shown. GDELT experiments are described in Section 5.5 followed by discussion of results in Section 5.6.

5.1. *Experiment Design*

Data Sources: Twitter and GDELT data were used for experiments. Twitter dataset is composed of approximately 60GB of Twitter data obtained from Twitter streaming API with over 15,000,000 tweets related to *civil unrest* in South America for time period September 2012 to April 2013. The GDELT experiments were performed using GDELT reduced eventset data from 1979-2014.
Environment: The computations are performed using Amazon AWS Elastic MapReduce with the data files in Amazon S3 data store as text files. The cluster consists of 2 data nodes for small, 4 for medium and 8 for large cluster. Each cluster node is of size large. Each cluster also has a large node as master orchestrating the work between data nodes. Each large node has 4 mappers and 2 reducers running on them. The worker nodes are controlled by master node. For any custom partitioners 4, 8 and 16 reducers are assumed. Custom partitioners allows for sending data to same reducer in *inverted value join* after shuffle for the distributed geocoding and entity network distance computations and not overly increasing size of data structures in memory. The sequential experiments are performed on a machine with 12GB RAM and a quad core Intel i7 processor. MapReduce code is in Java and the sequential operations are also performed using java 7 code. An overview of the distributed system architecture is shown in Figure 3. The distributed application runs on a cluster on Amazon Web Services (AWS). MapReduce jobs are run on AWS Elastic MapReduce (EMR) and data is read from and written to S3 buckets similar to distributed file system HDFS. Cluster uses Hadoop 2.5.1 and MapReduce2.
Comparison Methods: The flow of experiments is shown in Figure 7. The distributed algorithms do not limit the number of storylines with parameters such as Ripleys k-function or top connected entities by ConceptRank as in sequential storytelling in Santos *et al.* (2013). Other parameters such as calculating ConceptRank based on relationship types between entities, selecting relationship between entities, dampening factor for ConceptRank and selecting a good time window for entities for building storylines are kept the same as in Santos *et al.* (2013). To provide deep understanding of performance of distributed algorithms against sequential techniques, the next subsections describe four sets of experiments. The distributed storyline generation in Section 5.2 gives results of end-to-end storylines generation from raw tweets. It then breaks the overall process into constituent steps starting with distributed extraction of entities using NLP libraries in standfordner (2013) in Twitter data and compares sequential results against distributed results on multiple data and cluster sizes to show scalability in processing increasing larger data sizes with larger clusters. It also performs the same comparisons of distributed

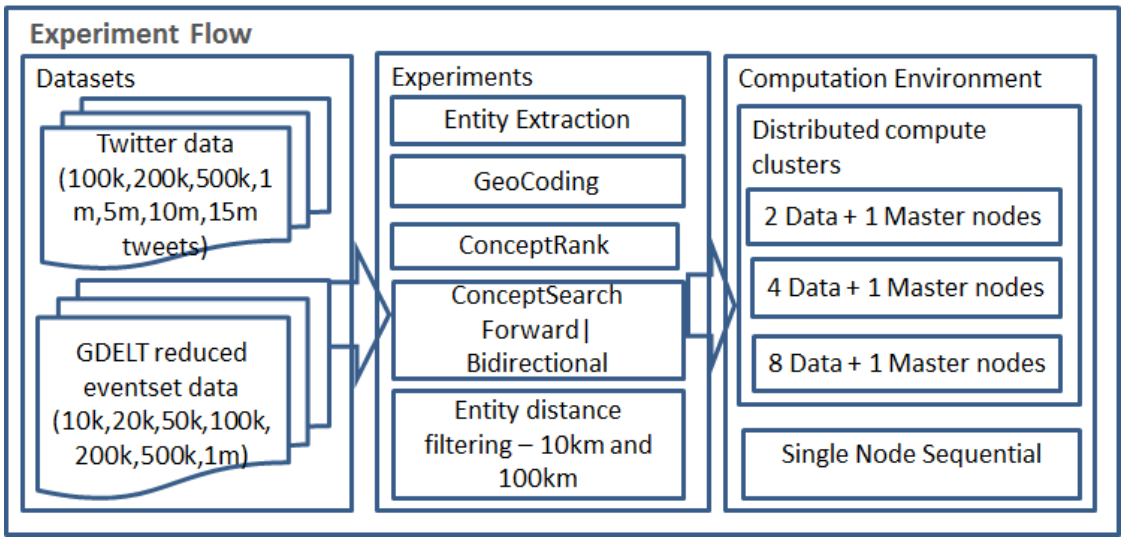

Figure 7.: Experiment flow for Twitter and GDELT tasks. Shows the various comparisons performed with sequential tasks and distributed tasks and overall performance incorporating multiple steps. Describes the various size datasets used in experiments.

geocoding using geonames geocoding api in GeoNames *et al.* (2015) and ConceptRank calculations against sequential implementations. The *ConceptSearch* graph traversal in Section 5.3 describes the bi-directional and forward only tree wise traversal of ConceptGraph to generate storylines on various data and cluster sizes. Results show small impact of using multiple starting points to perform storyline exploration instead of single preselected storyline starting point on performance and system scalability upon incremental demands on ConceptSearch. The entity network distance based filtering section describes the impact on reducing the number of storylines by filtering based on total distance between entity pairs. Thresholds of 10km and 100km coordinate distance between entity pairs is used in experiments. It shows the scalability of the filtering on multiple data and cluster sizes. The GDELT experiments in Section 5.5 discusses using the distributed platform to generate storylines using structured GDELT data. It compares GDELT distributed entity extraction against sequential results and then proceeds to show stability of GDELT distributed end-to-end performance and ability of entity distance filtering to limit number of storylines.

5.2. *Experiments with Twitter data*

The experiments with Twitter data based storytelling include the several distributed computations in steps. The comparisons of distributed operations and corresponding sequential ones are discussed in this section. The experiments for generating storylines from raw tweets including all the steps in generation are discussed first. Distributed storytelling generation includes distributed entity extraction, distributed ConceptRank computation, distributed ConceptSearch and entity network distance based filtering. An iterative ConceptSearch is performed of ConceptGraph with the starting point as starting point and creates all paths with nodes and edges from the starting node to all other connected nodes. Since these computations are performed using Ripleys k-function on a ConceptGraph in memory in sequential generation after user selects starting entity from a list, there are no exact comparisons with distributed implementation. Iterative

Table 2.: Twitter Storylines generated from sequential vs distributed approach. These storylines are all from starting point 'protest' for tweets collected in December 2014. They are sorted in order of ConceptRank and additional storylines generated in Distributed column are due to all storylines generated exhaustively in distributed paradigm.

	Sequential	**Distributed**
1	protest 1.7631 $\xrightarrow{mentions}$ #nyc still 1.4864 $\xrightarrow{going}$ strong 1.6992 $\xrightarrow{mentions}$ enforcement 1.5901	protest 1.7631 $\xrightarrow{mentions}$ #nyc still 1.4864 $\xrightarrow{going}$ strong 1.6992 $\xrightarrow{mentions}$ enforcement 1.5901
2	protest 1.7631 $\xrightarrow{mentions}$ #nyc still 1.4864 $\xrightarrow{going}$ strong 1.6992 $\xrightarrow{mentions}$ myanmar 1.6537	protest 1.7631 $\xrightarrow{mentions}$ #nyc still 1.4864 $\xrightarrow{going}$ strong 1.6992 $\xrightarrow{mentions}$ myanmar 1.6537
3	protest 1.7631 $\xrightarrow{mentions}$ #nyc still 1.4864 $\xrightarrow{going}$ strong 1.6992 $\xrightarrow{unites}$ malaria 1.5901	protest 1.7631 $\xrightarrow{mentions}$ #nyc still 1.4864 $\xrightarrow{going}$ strong 1.6992 $\xrightarrow{unites}$ malaria 1.5901
4		protest 1.7631 $\xrightarrow{mentions}$ #nyc still 1.4864 $\xrightarrow{going}$ strong 1.6992 $\xrightarrow{descends}$ parish council meeting 1.4864
5		protest 1.7631 $\xrightarrow{mentions}$ #nyc still 1.4864 $\xrightarrow{going}$ strong 1.6992 $\xrightarrow{mentions}$ feministabulous 1.4864

ConceptGraph search based storyline generation is highly computationally intensive and produces a lot more storylines than sequential ConceptGraph traversal. The final performance results of the entire distributed storyline generation algorithm with varying cluster and data sizes is shown in Figure 8. As the results clearly show, distribution does allow for scaling of Storytelling steps and makes the end to end process run in scalable and more robust way providing deeper insights. The starting point 'Mexico' was used in these experiments.

We ran distributed storytelling against several interesting tweet datasets. Table 2 shows some of the storylines created from Twitter data extracted with keywords 'protest' and 'demonstration' for January 2015. The two columns show storylines that would be generated in distributed computation that would otherwise be dropped in sequential processing due to slightly lower ConceptRank of the last entity in storyline even though these storylines would likely be ones analyst would like to dig deeper into. The relationship 'mentions' is used when there is no verb enumerating relationship between entities. These storylines show protest activities in NYC over law enforcement, Myanmar and impacting parish council meeting. All these storylines need to be evaluated by analysts instead of only a few based on ConceptRank.

5.2.1. Distributed Entity extraction

Performance of distributed entity extraction using key-value pair based operations proposed in previous sections is shown in Figure 9. Applying the distributed entity extraction technique clearly shows the parallel scalable nature of the problem. Using the nlp package and a classifier, extracting entities from each individual tweet in parallel shows remarkable performance improvement over sequential extraction. The sequential execution does not even complete for 15 million tweets dataset as the program runs out of memory. The initial overhead in starting the mapreduce job is the reason it starts out taking longer for smaller number of tweets but then scales well for very large number of tweets to the extent that a 2 node cluster is able to comfortably complete distributed extraction for the largest 15 million tweet set.

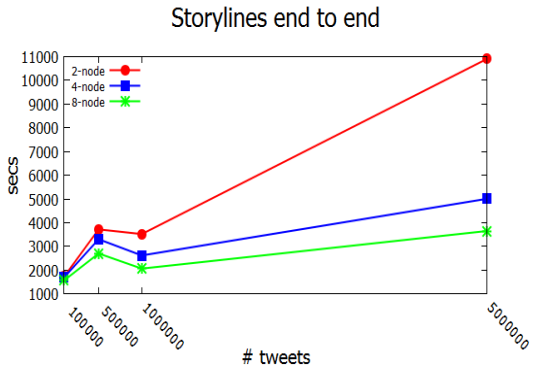

Figure 8.: Performance of distributed Storylines generation process combining various distributed tasks.

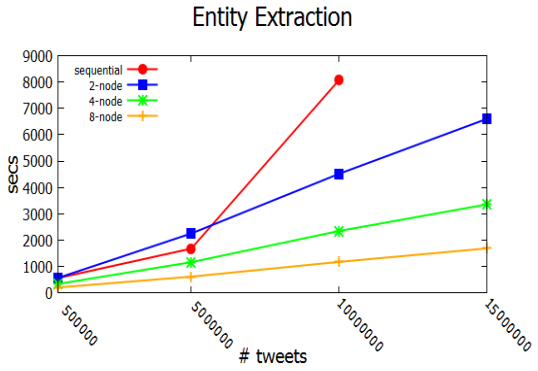

Figure 9.: Distributed entity extraction comparisons against sequential. Distributed experiments were performed on multiple size clusters.

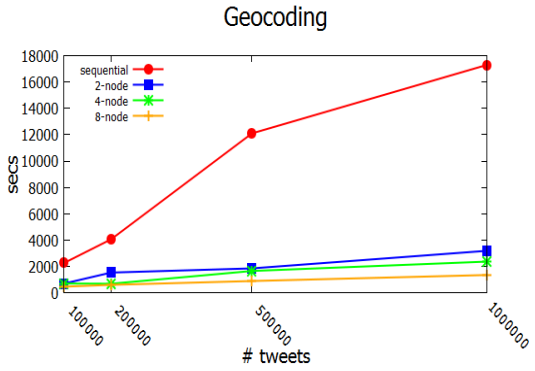

Figure 10.: Distributed and sequential performance of fetching geo coordinates of tweet locations from *Twitter* metadata.

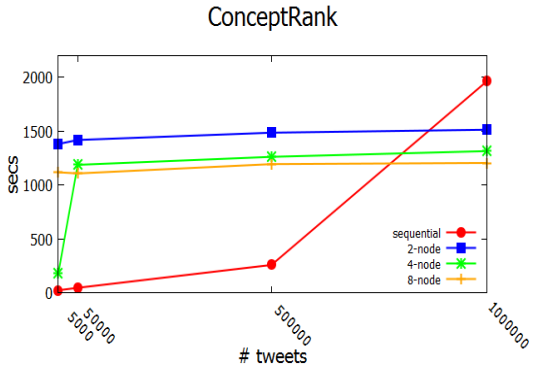

Figure 11.: Performance of distributed ConceptRank calculations compared to Sequential.

5.2.2. Distributed Entity GeoCoding

It is a lot more efficient to use distributed computing paradigm to geo-encode each entity location once and apply it to all the entities which were found at that location. This drastically cuts down on the computing and financial cost for calling the web based *geonames* geo-coding api and associating geo coordinates for the location. The improved performance of distributed geocoding is shown in Figure 10. As large part of the task is network requests, performance of various cluster sizes is similar and significantly better than sequential geocoding.

5.2.3. Distributed ConceptRank calculation

Computing ConceptRank of storylines is key to sorting them. The result of ConceptRank calculation on different datasets is shown in Figure 11. Distributed ConceptRank calculation using Shimmy pattern in Lin and Schatz (2010) is iterative and inherently different from the ConceptRank calculation on a single node. Sequential ConceptRank traverses ConceptGraph in memory and iteratively computes ConceptRank of each node while distributed ConceptRank partitions the graph and iteratively updates ConceptRank in each partition hence being highly scalable. The results in Figures 11 clearly

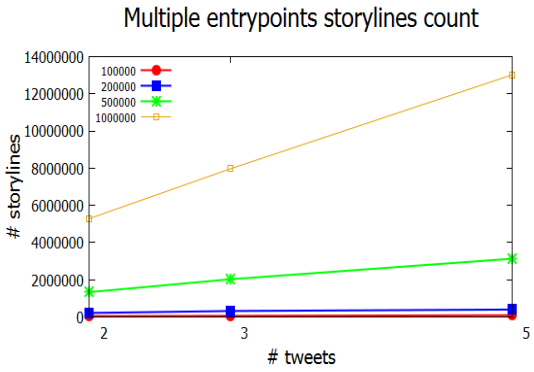

Figure 12.: Number of storylines created by multiple starting points. Bi-directional ConceptSearch is used.

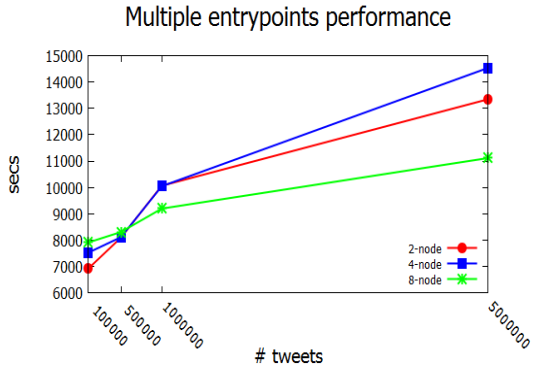

Figure 13.: Performance with 5 starting points ConceptSearch. Starting points are 'Brazil','Mexico', 'Gracias', 'Caro' and 'Photo'.

highlight the scalability of ConceptRank calculation based on *Shimmy* paradigm over sequential ConceptRank calculation. This performance gap become even more pronounced with larger graphs which invariably happen with dense entities and relationships and makes it imperative to use the distributed approach. The results are less pronounced than entity extraction because of the iterative nature of the distributed solutions but the distribution pays off in a major way when the size of graph increases. Since Shimmy algorithm first partitions data and then performs ConceptRank computation on each partition in parallel, the performance of 2 node and 4 node clusters for smaller data sets is worse compared to sequential execution due to the overhead of starting 3 mapreduce jobs involved, but with increasing number of nodes and increasing size of clusters the advantage of distribution started to show clearly.

5.3. *Distributed ConceptSearch*

In this experiment, ConceptGraph traversal is performed using the distributed *ConceptSearch* algorithm. Each iteration generates storylines one level deeper and is a separate MapReduce task. The results in Figure 14 show performance of *ConceptSearch* that generated storylines 4 entities long. The experiments are considered successful if the set of storylines found with sequential execution are found with parallel algorithm as well along with others. The performance results on tweets is shown in Figure 14. Bi-directional ConceptSearch scales well with increasing data sizes for each cluster. The dataset is composed of tweets from which a ConceptGraph of entities and relationships was generated. Cluster size is incrementally increased to compare parallel performance of algorithms on increasing data sizes.

The result of forward and bi-directional traversals and the number of storylines generated is shown in Figure 15. With bi-directional ConceptSearch as shown in Figure 15 the number of storylines increases significantly, the large dataset of 5m tweets produces over 22 million storylines. Switching to forward ConceptSearch can be useful in containing the number of storylines to a few hundred thousand in this case. The starting point 'Brazil' was used in these experiments.

The results for performance of forward traversal are given in Figure 17 and Figure 16 shows storylines count for forward traversal. The performance for this dataset indicates effectiveness of parallelization on processing graph data after extraction of semantic

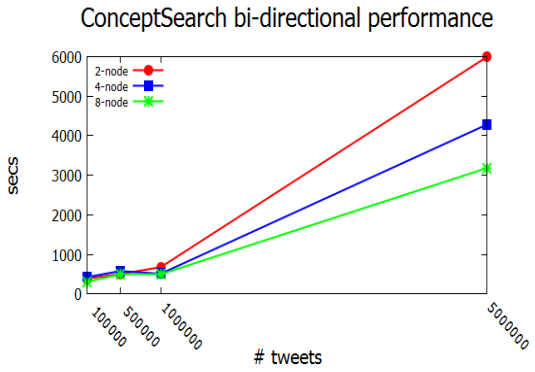

Figure 14.: Performance of distributed bi-directional ConceptSearch. It scales well with increasing cluster and data sizes.

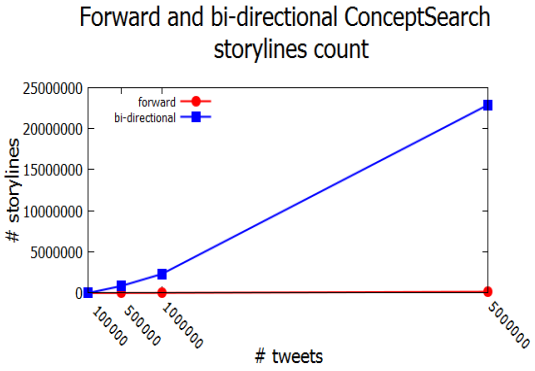

Figure 15.: Number of storylines forward vs bi-directional ConceptSearch. Shows the effectiveness in containing storylines count with forward ConceptSearch.

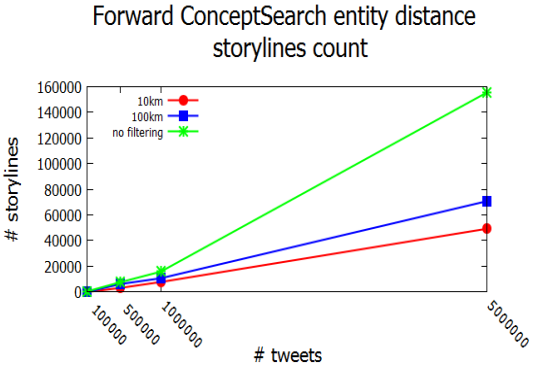

Figure 16.: Storylines count forward ConceptSearch with entity network distance based filtering. Combination of forward traversal which is stricter in storylines generated along with entity distance filtering limits storylines significantly.

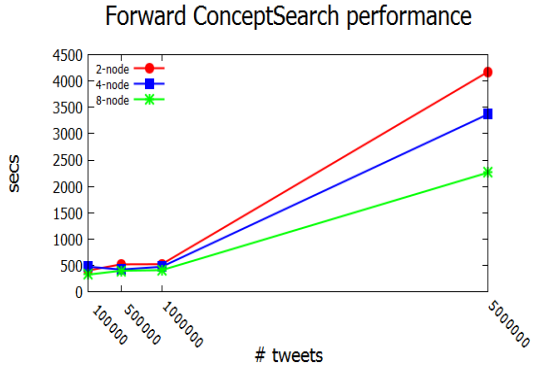

Figure 17.: Performance of distributed forward ConceptGraph traversal. Forward traversal comptation and space complexity is similar to bi-directional.

keywords from text and then building storylines from starting points. The distribution techniques do not pay off with smaller datasets as there is some overhead involved in using a distributed framework to process the task as subtasks but with increasing data size the paradigm becomes effective and at some stage becomes the only way to perform storytelling beyond a certain data size.

ConceptSearch supports incremental provision of multiple starting points as most of the storyline generation process remains independent of starting points and in each ConceptGraph traversal iteration multiple starting points can be processed for respective storylines with little incremental cost to MapReduce processes. The 5 starting points used in experiments are "Brazil, Mexico, Gracias, Cara and Photo". We run experiments with first 2, first 3 and all 5 starting points and show the relative stability of the storyline generation flow and the ConceptSearch expense. The number of storylines generated with multiple starting points are shown in Figure 12. The performance of the multiple

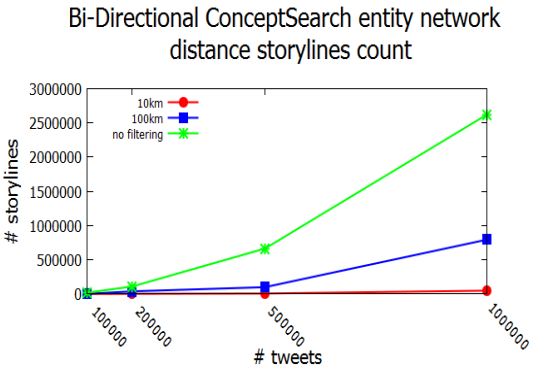

Figure 18.: Reducing the number of storylines by entity network distance filtering. This allows analyst to focus on sections of total possible storylines.

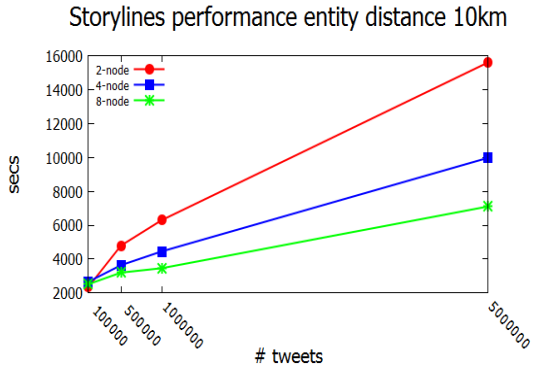

Figure 19.: Distribution of storylines generation based on pairwise distance between all entities between storylines distance computation. Performance changes little from one threshold to another. Distance is chosen as 10km for these experiments.

starting points under different dataset sizes and different cluster sizes for all the 5 starting points is shown in Figure 13. It clearly shows the stability of performance of storyline generation flow and its key component ConceptSearch under multiple starting points.

5.4. *Entity Network distance filtering*

In storylines entity network distance experiments, the efficacy of lowering the number of storylines using distances between entity pairs calculated in a distributed fashion is shown. Experiments with limiting the number of storylines is conducted by calculating pairwise distances of entities and combining for filtering based on the total length between entities in sequence. Domain experts have observed that stories with long distance length between entities tend to be less important than ones with lesser distance. Hence the focus on calculating the distances between entity pairs and keeping the storylines with lower edge pairwise distance from start to end node for evaluation and filtering out the rest. The decrease in number of storylines with lower entity network distance thresholds are shown in Figure 18. It clearly shows the marked reduction in number of storylines once we limit the threshold to 100km and even more significant reduction when lowering the threshold to 10km. The performance impact of applying filtering is shown in Figure 19. It shows performance of storylines generation along with entity distance based filtering for threshold of 10km and the improvement in performance with increase in cluster size is consistent with storylines generation experiment proving that additional filtering does not significantly add to processing and is linearly scalable with additional nodes. In addition Figure 16 shows the combination of entity network distance filtering with forward ConceptSearch for limiting storylines generation provides the maximum filtering in order for an analyst to start with the most contextually and spatially relevant storylines and work their way outwards.

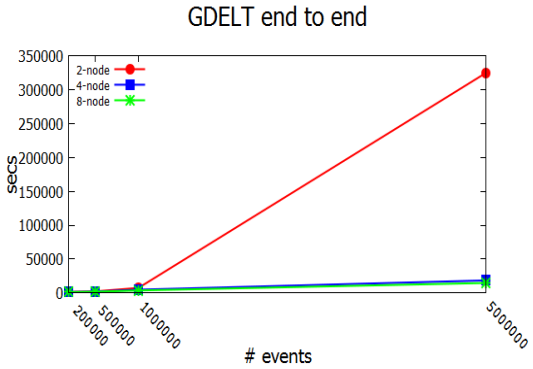

Figure 20.: Distribution of GDELT storylines end to end computation. The data is structured and requires fewer processing steps before storylines are calculated compared to Twitter data.

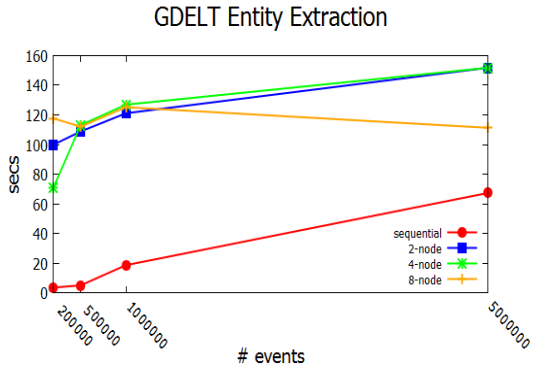

Figure 21.: Distribution of GDELT raw data processing and entity and relationship extraction. Shows due to structured nature of GDELT events the advantages in distributed paradigm do now show up until volume of data parsed increases significantly.

5.5. *Experiments with GDELT data*

Storylines generation against GDELT data is performed to ensure relevant storylines are generated using more structured data sets. The Global Database of Events, Language, and Tone (GDELT) is a project that crawls various data sources around the world including broadcast, print and online news sources to capture significant events along with other information such as geographical locations, actors, time etc. The format of the data is a csv file with fields such as *eventid, actorname, actorcountry, actorrelegion, actiongeolat, actiongeolong* etc. Since the data is inherently structured, it is extremely efficient to extract entities and relationships from it. The efficiencies are accentuated further with each event being self contained and no geocoding needed to get coordinates of locations. The ConceptRank and ConceptSearch algorithms are performed in the same way as Twitter data.

Figure 20 shows the vast performance improvements in processing GDELT data inherently taking advantage of its structured nature compared to more unstructured text processing in tweets. Text parsing based entity extraction and location geocoding and combining entities with their geo-coordinates using *inverted value join* need not be performed. That also means importance of distributed technique is less for GDELT data parsing than Twitter. However for much larger datasets the distribution advantage begins to reassert itself. Figure 21 shows the scalability of extracting entities from structured GDELT events. Figure 22 shows the performance of GDELT storytelling when applying entity network distance thresholds while Figure 23 shows the decrease in storylines when the thresholds are applied to aid analysts. Since coordinates are available for each event in each structured record in GDELT data the entity network distance based filtering is more computationally intensive. Table 3 shows some of the storylines created from GDELT data. Here looking up the CAMEO codes for actors and events from GDELT metadata in Leetaru and Schrodt (2013), ABW stands for Aruba, AUSGOV for Australian Government, NLD for Netherlands, BEL for Belgium, AUSCOP is Australian police force, AUSOPP is Australian opposition and AUSEDU is Australian education. For the event codes 010 is code for 'Make Statement', 091 is 'Investigate crime, corruption', 112 is 'Accuse', 190 is 'Use conventional military, force', 036 is 'Express intent to

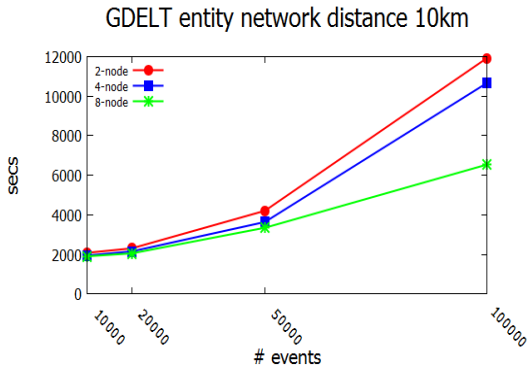

Figure 22.: Entity Network distance based filtering for GDELT. The performance of storylines generation remains stable with the additional filtering step.

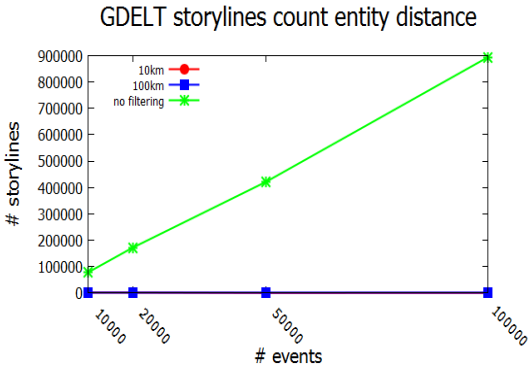

Figure 23.: Number of storylines at varying entity network distance thresholds. Shows the effectiveness in containing number of storylines allowing analyst to focus.

Table 3.: GDELT Storylines generated from sequential vs distributed approach. The distributed column shows the additional storylines that top k ConceptRank based traversal in sequential implementation will not generate. Analysts find additional storylines important.

	Sequential	**Distributed**
1	abw 1.4317 $\xrightarrow{010}$ nld 2.0534 $\xrightarrow{112}$ ausgov 1.6697 $\xrightarrow{091}$ auscop 1.7121	abw 1.4317 $\xrightarrow{010}$ nld 2.0534 $\xrightarrow{112}$ ausgov 1.6697 $\xrightarrow{091}$ auscop 1.7121
2	abw 1.4317 $\xrightarrow{010}$ nld 2.0534 $\xrightarrow{112}$ ausgov 1.6697 $\xrightarrow{190}$ ausopp 1.7121	abw 1.4317 $\xrightarrow{010}$ nld 2.0534 $\xrightarrow{112}$ ausgov 1.6697 $\xrightarrow{190}$ ausopp 1.7121
3		abw 1.4317 $\xrightarrow{010}$ nld 2.0534 $\xrightarrow{112}$ ausgov 1.6697 $\xrightarrow{036}$ ausedu 1.4579
4		abw 1.4317 $\xrightarrow{010}$nld 2.0534 $\xrightarrow{112}$ ausgov 1.6697 $\xrightarrow{030}$ bel 1.3948

meet or negotiate' and 030 is for 'Express intent to cooperate'. These storylines show interaction between Australian government and Netherlands over activities in Aruba.

5.6. *Discussion*

This subsection discusses the results shown in charts earlier in the section. The results show the efficiency and effectiveness of bi-directional and forward ConceptSearch to build storylines relevant to the subject domain in large volumes. They also show the effectiveness of filtering storylines by entity network distance to allow analyst to focus on smaller set of spatially connected storylines.

Location based filtering: The large number of storylines with distributed technique also allows them to find the connections not previously possible. However, due to large number of storylines produced by distributed technique, to discover stories of interest it is imperative that the search space be limited by location. For example, the concepts discussed earlier in section 5 (*demonstrations, protest*) are general, until they are combined with locations specific to the *NYC* protests. If not bounded by the location radius, the semantic constraint *protest* could generate stories for the NYC protests, Ferguson protests, or any others around the country making it difficult for analyst to sift through

the large number of storylines generated. Hence, the importance of the spatial component of our proposed work to narrow down the number for storylines to specific events.

Impact of bi-directional and forward ConceptSearch on storylines: The storylines generated connect entities across tweets hence are very powerful. The assumption of starting with the first entity of each tweet as having outgoing links to other entities in tweets is weak and there would be times when even in short text of tweets the key entity in a tweet will be a following entity after first. That makes bi-directional search even though marginally more expensive compared to forward ConceptSearch important. However due to increase in analysis complexity when all potential stories are generated using bi-drectional ConceptSearch, the filtering applied to storylines by forward ConceptSearch allows analyst to not have to search exhaustively through storylines generated by any entity in a tweet but focus on ones that propagate via first tweet entity and refine the analysis on storylines. To deal with the sheer volumes of tweets generated from bi-directional search the entity network distance based filtering becomes even more important. From 5 million tweets in Figure 15 experiments over 20 million storylines are generated which will be overwhelming for any analyst. Hence the various ways of filtering storylines and focus on subsets of them becomes crucial to take full advantage of the opportunities offered by distributed storytelling without the overwhelming number of storylines making the tool of little value.

GDELT validation: GDELT results also validate the same conclusions that Twitter results provide regarding scaling of storylines generation from GDELT data in parallel and the effectiveness of entity network distance based filtering on generated storylines. The structured nature of data makes entity extraction easier hence less impact of distributed entity extraction in scaling yet the remainder of storytelling steps such as ConceptRank calculation and ConceptSearch being same as Twitter makes use of distributed technique critical. Availability of coordinate locations of all actors in each event makes entity distance filtering more effective and relevant.

6. Conclusion

In this paper we proposed a technique for exploring interactions from spatio-temporal dynamic real-world storylines generated from microblog and event data sources and performed at scale using distributed computing paradigm. Our approach distributes multiple step storytelling that includes entity extraction, geocoding, ConceptGraph creation and traversal for storylines generation and entity network distance based filtering of storylines. Novel techniques such as *ConceptSearch* perform graph traversal operations at scale and *inverted value join* allow disparate sets of values to be combined together efficiently in key-value pair paradigm. Experiments on *Twitter* and GDELT data sources show the power of distribution techniques in scaling computations and usefulness of storylines generated. With the use of distributed approach, analysis does not need to limit itself to much smaller number of entities and storyline generated from raw Twitter data in sequential storytelling. Examples of storylines generated demonstrate the high potential for exploratory analysis using current events such as the *NYC protests*, *Ferguson protests* and the *Charlie Hebdo attacks*. The deeper insights from results generated by distributed approach are significant improvement over limited insights from smaller output by sequential approach.

References

Afrati, F.N. and Ullman, J.D., 2010. Optimizing Joins in a Map-Reduce Environment. *In*: *P13th International Conference on Extending Database Technology*, EDBT'10, Lausanne, Switzerland New York, NY, USA: ACM, 99–110.

Apache and Hadoop, 2014. http://hadoop.apache.org. *Apache Hadoop*.

Bulu, A. and Madduri, K., 2012. Graph partitioning for scalable distributed graph. *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, 1 (1), 793–803.

Chan, J., Bailey, J., and Leckie, C., 2008. Discovering correlated spatio-temporal changes in evolving graphs. *Knowledge and Information Systems*, 16, 53–96.

Chan, J., Bailey, J., and Leckie, C., 2009. Using graph partitioning to discover regions of correlated spatio-temporal change in evolving graphs. *Intelligent Data Analysis*, 13, 755–793.

Dean, J. and Ghemawat, S., 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51 (1), 107–113.

Finin, T., *et al.*, 2010. Annotating Named Entities in Twitter Data with Crowdsourcing. *In*: *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, CSLDAMT '10, Los Angeles, California Stroudsburg, PA, USA: Association for Computational Linguistics, 80–88.

GeoNames, Geocoding, and API, 2015. http://www.geonames.org/. *GeoNames*.

George, B., Kang, J., and Shekhar, S., 2009. Spatio-temporal sensor graphs (STSG): A data model for the discovery of spatio-temporal patterns. *IDA*, 13, 457–475.

Hossain, M.S., *et al.*, 2011. Helping Intelligence Analysts Make Connections. *In*: *Workshop on Scalable Integration of Analytics and Visualization*, AAAI '11, 22–31.

Hossain, M.S., *et al.*, 2012a. Stortytelling in Entity Networks to Support Intelligence Analysts. *In*: *KDD'12*, 1375–1383.

Hossain, M.S., *et al.*, 2012b. Connecting the Dots between PubMed Abstracts. *PLoS ONE*, 7 (1).

Kumar, D., *et al.*, 2008. Algorithms for Storytelling. *IEEE TKDE*, 20 (6), 736–751.

Leetaru, K. and Schrodt, P.A., 2013. GDELT: Global data on events, location, and tone, 19792012. *ISA Annual Convention*, 2.

Lin, J. and Schatz, M., 2010. Design Patterns for Efficient Graph Algorithms in MapReduce. *In*: *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*, MLG '10, Washington, D.C. New York, NY, USA: ACM, 78–85.

Merrill, D., Garland, M., and Grimshaw, A., 2012. Scalable GPU Graph Traversal. *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, 1 (1), 793–803.

Michelson, M. and Macskassy, S.A., 2010. Discovering Users' Topics of Interest on Twitter: A First Look. *In*: *Proceedings of the Fourth Workshop on Analytics for Noisy Unstructured Text Data*, AND '10, Toronto, ON, Canada New York, NY, USA: ACM, 73–80.

Mondo, G.D., *et al.*, 2013. Modeling consistency of spatio-temporal graphs. *Data and Knowledge Eng.*, 84, 59–80.

Osterman, A., Benedii, L., and Ritoa, P., 2014. An IO-efficient parallel implementation of an R2 viewshed algorithm for large terrain maps on a CUDA GPU. *International Journal of Geographical Information Science*, 28 (11), 2304–2327.

Qin, C.Z., *et al.*, 2014. A strategy for raster-based geocomputation under different parallel computing platforms. *International Journal of Geographical Information Science*, 28

(11), 2127–2144.

Santos, R.D., Boedihardjo, A., and Lu, C.T., 2012. Towards ontological similarity for spatial hierarchies. *In*: *ACM GIS QUeST*, 26–33.

Santos, R.D., *et al.*, 2013. Spatio-temporal Storytelling on Twitter. *Computer Science Technical Reports, Virginia Tech,https://vtechworks.lib.vt.edu/handle/10919/24701*.

Santos, R.D., *et al.*, 2014. Forecasting Location-based Events with Spatio-temporal Storytelling. *In*: *ACM SIGSPATIAL LBSN*, Dallas, Texas, USA.

Sarma, A.D., *et al.*, 2012. Fast Distributed PageRank Computation. *CoRR*, abs/1208.3071.

Shahaf, D. and Guestrin, C., 2012. Connecting Two (or Less) Dots: Discovering Structure in News Articles. *ACM Trans. Knowl. Discov. Data*, 5 (4), 24:1–24:31.

Singh, S., *et al.*, 2011. Large-scale cross-document coreference using distributed inference and hierarchical models. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 1 (1), 793–803.

standfordner, 2013. http://nlp.stanford.edu/software/CRF-NER.shtml. .

Turner, S., 1994. *In*: *The creative process: A computer model of storytelling and creativity.*, 122–123 Psychology Press.

Weitzel, L., Quaresma, P., and de Oliveira, J.P.M., 2012. Measuring Node Importance on Twitter Microblogging. *In*: *Proceedings of the 2Nd International Conference on Web Intelligence, Mining and Semantics*, WIMS '12, Craiova, Romania New York, NY, USA: ACM, 11:1–11:7.