Tornado: A Distributed Spatio-Textual Stream Processing System*

Ahmed R. Mahmood[1], Ahmed M. Aly[1], Thamir Qadah[1], El Kindi Rezig[1], Anas Daghistani[1], Amgad Madkour[1], Ahmed S. Abdelhamid[1], Mohamed S. Hassan[1], Walid G. Aref[1], Saleh Basalamah[2]
[1]Purdue University, West Lafayette, IN [2]Umm Al-Qura University, Makkah, KSA
[1]{amahmoo, aaly, erezig, amgad, samy, msaberab, aref}@cs.purdue.edu,
[1]{tqadah, anas}@purdue.edu, [2]smbasalamah@uqu.edu.sa

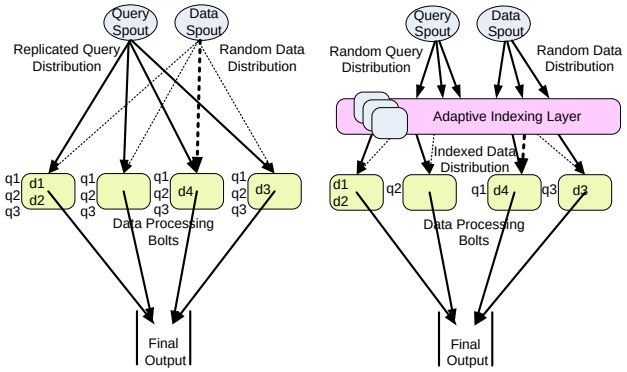
ABSTRACT

The widespread use of location-aware devices together with the increased popularity of micro-blogging applications (e.g., Twitter) led to the creation of large streams of spatio-textual data. In order to serve real-time applications, the processing of these large-scale spatio-textual streams needs to be distributed. However, existing distributed stream processing systems (e.g., Spark and Storm) are not optimized for spatial/textual content. In this demonstration, we introduce Tornado, a distributed in-memory spatio-textual stream processing server that extends Storm. To efficiently process spatio-textual streams, Tornado introduces a spatio-textual indexing layer to the architecture of Storm. The indexing layer is adaptive, i.e., dynamically re-distributes the processing across the system according to changes in the data distribution and/or query workload. In addition to keywords, higher-level textual concepts are identified and are semantically matched against spatio-textual queries. Tornado provides data deduplication and fusion to eliminate redundant textual data. We demonstrate a prototype of Tornado running against real Twitter streams, where the users can register continuous or snapshot spatio-textual queries using a map-assisted query-interface.

1. INTRODUCTION

The widespread use of GPS-enabled cellular devices and the increasing popularity of micro-blogging platforms (e.g., Twitter) has resulted in generating large volumes of geo-tagged data (e.g., a tweet has both spatial and textual attributes).

Consider the following scenario. A tourist is visiting a city for the first time and wishes to know how people currently visiting the attractions feel about them in order to choose the best attraction to visit (i.e., avoid busy, uninteresting, or closed attractions). The tourist submits a query to a real-time spatio-textual system that finds tweets originating in the vicinity of an attraction and that are discussing the attraction. Tweets arrive at high rate and the data sources containing information about attractions are large (e.g., OpenStreetMap is about 300 GB). Processing these spatio-textual queries calls for a scalable and distributed real-time spatio-textual query processing system that can efficiently process and analyze these spatio-textual data streams in real-time.



(a) Storm topology (b) Tornado topology

Figure 1: A Storm topology vs. a Tornado topology.

Although being in the big data era, existing platforms are not optimized for spatio-textual query processing over big data streams. Existing platforms fall into one of the following categories: (1) distributed batch systems (e.g., [1, 6, 7]) that are not optimized for spatio-textual query processing and that suffer from high latency, (2) distributed spatio-textual batch systems (e.g., [8]) that can handle spatio-textual queries yet they suffer from high latency, (3) distributed main-memory and streaming systems (e.g., [3, 4, 12]) that are also not optimized for spatio-textual query execution, (4) centralized stream-based spatio-textual systems (e.g., [9]) that are constrained by the resources of the single machine they run on.

This demonstration presents Tornado, a distributed system for real-time processing of spatio-textual queries over data streams. Tornado extends Storm that is a distributed and fault-tolerant general-purpose stream processing system [4]. Stream processing in Storm is implemented using three main components; *spouts*, *bolts*, and *topologies*. A spout is a source of input data streams. A bolt is a data processing unit. A topology is a directed graph that connects spouts and bolts to form a stream processing pipeline. Tornado extends Storm with an adaptive indexing layer that improves the query processing performance of spatio-textual queries.

Figure 1 contrasts a Storm topology and a Tornado topology when processing queries with spatio-textual predicates. As in Fig-

*This research was supported in part by National Science Foundation under Grants IIS 1117766 and IIS 0964639.

Proceedings of the VLDB Endowment, Vol. 8, No. 12

ure 1, Storm has no notion of spatial locality, and hence queries are replicated across all bolts. This imposes network overhead and implies overloaded bolts. In contrast, Tornado uses an adaptive indexing layer (Figure 1(b)) that ensures that queries are not replicated and that the data is sent only to the relevant bolts.

Tornado provides semantic search capabilities that go beyond conventional keyword-based matching. We identify and use concepts over streaming data in an online fashion in order to determine how semantically related the identified concepts are to the spatio-textual queries. For example, consider the query that finds all the nearby points of interest that are related to *Pollution*. Assume that there is a nearby factory that is semantically related to the concept *Pollution* with a specific score. Then, the factory is reported as a result of the query if the relatedness score is above a certain threshold.

We demonstrate Tornado with its real-time spatio-textual processing capabilities using real annotated map data from OpenStreetMap [2], Tweets arriving from Twitter fire-hose, and synthetic trajectory data from the Minnesota Traffic Generator [11].

2. OVERVIEW OF TORNADO

2.1 Features

The main features of Tornado are summarized as follows:

- **Distributed Main-Memory Spatio-Textual Indexing.** This index ensures that data that is relevant to a specific query is sent to the same processing bolt where the query resides. Realizing this index without introducing system bottlenecks is an important design challenge.

- **Adaptivity to Skewed Query-Workload and Data Distribution.** Because of the dynamic nature of the spatio-textual data, data distribution may change over time. Furthermore, typical spatial query-workloads exhibit skewed access patterns, where certain spatial regions receive queries more frequently than others. To address these challenges, Tornado realizes a novel adaptive and query-workload awar data partitioning mechanism. An important characteristic of this partitioning mechanism is that it does not assume prior knowledge of the query-workload or the data distribution. Tornado maintains statistics about the data distribution and the number of queries received at each bolt. Whenever the distribution of the data or the query-workload changes, Tornado reacts by incrementally updating the layout of the bolts in the underlying topology.

- **Support for Spatio-textual Trajectories with Limited Histories.** Tornado is able to efficiently maintain limited histories (i.e., sliding windows) of spatio-textual trajectories, e.g., trajectories of moving objects for the past one day, three hours, etc [10]. A spatio-textual trajectory is the history of locations visited by a moving entity. This history of locations is tagged with any textual content that is produced at any specific location (if any). Maintaining limited trajectories allows performing several important analytical queries (e.g., traffic analysis) without exhausting memory resources by not keeping the entire history in memory. These queries cannot be answered if we only keep current-time location data.

- **Spatio-textual Querying.** Nowadays, many complex spatio-textual operators and their matching complex indexing structures are commonly being proposed in the literature. However, in the authors' opinion, this approach is against the spirit of SQL and relational algebra. In relational algebra, simple relational operators are offered, e.g., relational selects, projects, and joins, that are composable to form more complex queries. Tornado's query language follows the philosophy of SQL and relational algebra in the following sense. Tornado offers simple declarative spatial, textual, and semantic building block operators and predicates that are composable to form complex spatio-textual queries.

- **Deduplication and Fusion.** Tornado identifies duplicate spatio-textual data, e.g., tweets that convey the same information, and returns a single representation of these duplicates (fusion). The benefit of this phase is twofold. On one hand, it reduces the number of tweets to be processed by the query evaluators, hence, improving the query response time. On the other hand, it diversifies the content of the returned answer, and hence improving the quality of the query results.

- **Map-assisted Query Interface.** Users of Tornado can express spatio-textual queries using an SQL-like query language that is coupled with a map. For example, users can specify a set of focal points by clicking on the map and reference these points in their k-nearest neighbor queries. This interface is different from [9], as Tornado provides an SQL-Like language to express user queries while [9] uses a web-based form to specify the various predicates.

- **Semantic Search.** Tornado provides semantic search capabilities extending beyond conventional Boolean keyword matches. Introducing semantic search allows Tornado to find the semantically relevant items that match the query.

2.2 Data Model

Tornado processes tuple data from mulitple sources that are of the following format: $\{srcid, oid, (x, y), t, text\}$, where $srcid$ is the data source identifier, oid is the identifier of the object reporting the incoming tuple. (x, y), t, and $text$ are the spatial location of the centroid of the object, the timestamp, and the textual content of the tuple, respectively.

2.3 Supported Query Predicates

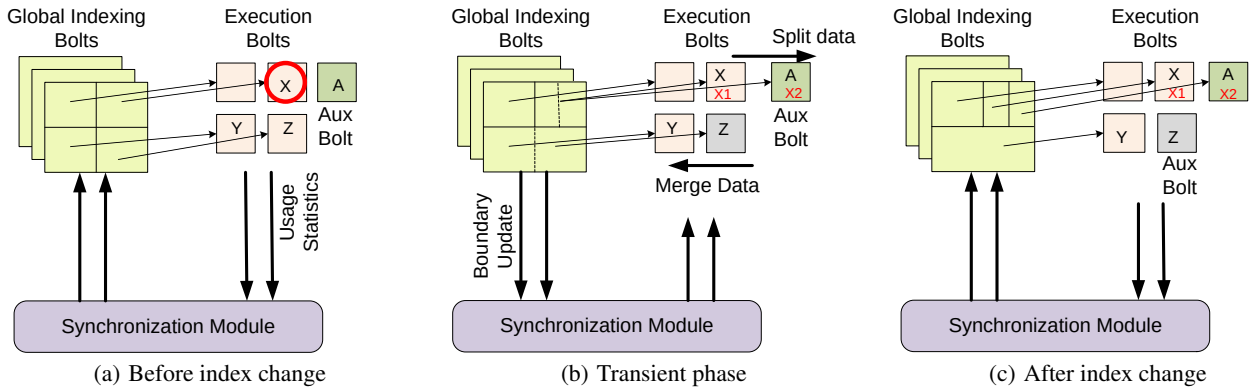
Tornado is meant to handle a wide range of spatio-textual queries. In the current phase of Tornado, we are mainly concerned with both snapshot and continuous versions of the following query predicates: *spatio-textual range and kNN select and join predicates* (including distance joins), *textual similarity and semantic matching predicates*, *limited history, and temporal predicates*. Examples of supported queries are given in Section 4.

3. SYSTEM ARCHITECTURE

In this section, we give an overview of the layered architecture of Tornado illustrated in Figure 2.

3.1 Adaptive Indexing Layer

Indexing in Tornado is distributed and is composed of (1) a global spatial index, and (2) local spatio-textual indexes. Figure 3 illustrates Tornado's indexing module. All incoming data and queries navigate though the global index to be assigned to a query processing unit (i.e., a bolt). To avoid performance bottlenecks, the global index is replicated across several bolts. A local spatio-textual index is composed of multiple in-memory k-d trees (one per data source). Each non-leaf node in the k-d tree is augmented with an inverted list that summarizes the textual contents



(a) Before index change (b) Transient phase (c) After index change

Figure 4: Index adaptivity in Tornado.

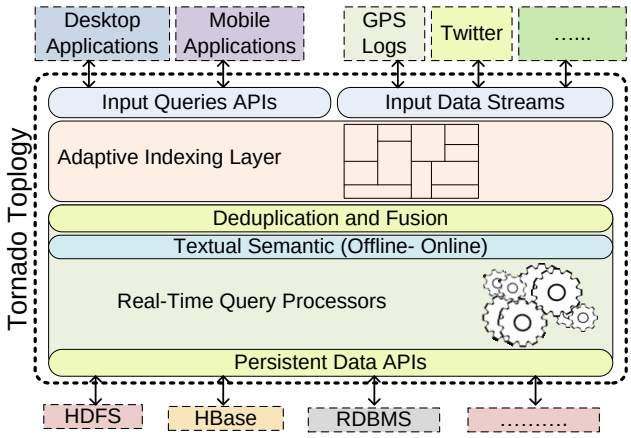


Figure 2: The layered architecture of Tornado.

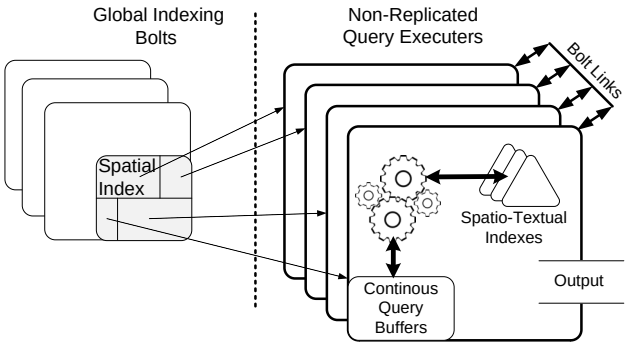


Figure 3: Indexing in Tornado.

of its child nodes. The inverted lists help speedup the processing of the spatio-textual queries.

It is expected that the system workload will not be the same at all times, and hence having a static indexing layer can result in poor system performance. Some processing bolts may get overloaded, while other processing bolts may get underutilized. Tornado is adaptive to changes in both the data and query workload. It is challenging to ensure workload adaptivity for the following reasons:

- **Replicated Index Bolts:** In Tornado, incoming tuples go to any instance of the indexing bolts. This mandates that all the indexing bolts maintain the same layout of the global index to ensure consistency when sending the tuples to the processing bolts.
- **No Global System View:** It is relatively easy to achieve adaptivity in a centralized system, where the workload statistics are centralized. This is not the case in Tornado, where the workload statistics are distribued, and the indexing bolts do not have access to the memories of each other.

Tornado uses Apache ZooKeeper [5], an open-source distributed configuration and synchronization service, to synchronize the changes in the global index bolt. Zookeeper stores usage statistics (i.e., the number of data objects and queries processed) from the data processing bolts. The index bolts access these usage statistics from the zookeeper to detect when a change in the index is needed.

We model the overhead (i.e., cost), say C, corresponding to a bolt, say b, as: $C(b) = P(b) \times Q(b)$, where $P(b)$ and $Q(b)$ are the number of data objects and queries in b, respectively. To update the layout of the bolts, we split the space corresponding to the bolt, say Hb, with the highest overhead. We use a free auxiliary bolt to hold a portion of the data of Hb. To keep the number of bolts constant in the topology, which is a physical constraint due to cluster resources, we merge two adjacent bolts that have the lowest overhead into one new bolt. Thus, the layout of the bolts always follows the structure of a k-d tree. Tornado avoids unnecessary splits/merges by integrating its cost model with the cost of shuffling the data between the bolts. Figure 4 describes the steps of adapting to workload changes. In Figure 4(a), Bolt X is under heavy load while Bolts Y and Z are underutilized. Figure 4(b) represents the transient phase, where the data in Bolt X is split into x_1 (remains at Bolt X) and x_2 (sent to the free auxiliary Bolt A). The data from the underutilized Bolt Z is transferred to Bolt Y. After the transient phase, Bolt Z becomes the free auxiliary bolt to be used in future adaptivity actions. Figure 4(c) gives the global index and execution bolts after the change.

3.2 Query Evaluation Layer

The data processing layer is the working horse of Tornado. This layer is composed of a set of interconnected data processing bolts (evaluators). Evaluators are responsible for query evaluation over streamed data. Incoming spatio-textual data and queries go through the global indexing layer to determine the relevant evaluator bolts

to be sent to. An evaluator bolt is composed of two main components, (1) continuous queries buffer, (2) local spatio-textual indexes. Continuous queries are held in a main-memory buffer until revoked. Incoming streamed data is checked against the continuous queries buffer to update the results of queries. The local spatio-textual index is composed of multiple k-d trees augmented with text as descried in Section3.1.

The following steps outline how the queries are evaluated in Tornado: (1) Check the query's spatial attributes against the global index. (2) Send the query to the relevant execution bolt(s). (3) Use the local spatio-textual index to evaluate the query. (4) Consult the neighboring bolts if needed, e.g., textual-kNN and textual distance join. (5) Report the query results. Continuous queries are held in a continuous-queries buffer and query results are updated based on the incoming data.

3.3 Data Input Layer

Tornado is able to ingest a wide range of spatio-textual data streams e.g., GPS logs of moving vehicles, tweets, and human interactions with online ads, as well as persistent (i.e., static) data, e.g., road networks and points of interests. Users of Tornado may submit queries from either desktop or mobile applications. In a Storm topology, input streams are accessed through spouts. Tornado extends Storm and follows the same conventions. In Tornado, there are three main types of inputs, namely (1) streamed queries, (2) streamed spatio-textual data, and (3) persistent data.

Tornado provides spouts for input types (1) and (2). For input type (3), we do not use spouts to read persistent data. To avoid the overhead of going though the indexing layer for persistent data, Tornado provides an API to directly load this data from disk storage, e.g., from HDFS, HBASE, or RDF stores, into the query processing bolts (i.e., the evaluators).

3.4 Deduplication and Fusion Layer

Tornado performs on-the-fly deduplication and fusion of spatio-textual data streams as the data arrives. For deduplication, we harness Tornado's spatial indexing scheme to consider only data objects that are geographically co-located. Incoming spatio-textual data objects are matched against previously processed streams. In case of high similarity, the incoming data object is deemed duplicate and is not returned. After identifying the duplicates, fusion is performed by returning the data that is deemed most informative.

3.5 Textual Semantics Layer

In Tornado, we compute the semantic relatedness among spatio-textual data to enrich the query results. The computation is performed in two phases: 1) an offline preprocessing phase, where we calculate semantic relatedness scores among keywords and concepts, and 2) an online phase, where we use the precomputed scores to match spatio-textual data with queries. If the overall semantic relatedness score between a data object and a query is above a certain threshold, the object is reported among the result set of the query.

4. DEMO SCENARIO

Real-time Human Experience: We demonstrate the scenario highlighted in Section 1, where a tourist issues a query to view the tweets that discuss attractions. The real-time feedback from tweeters can assist the tourist to avoid crowded or uninteresting attractions. This query can be expressed in Tornado as follows:

```
REGISTER QUERY q1 AS
SELECT * FROM OSM_Data AS O, Tweets AS T
WHERE WITHIN_DISTANCE(T, O, Threshold)
```

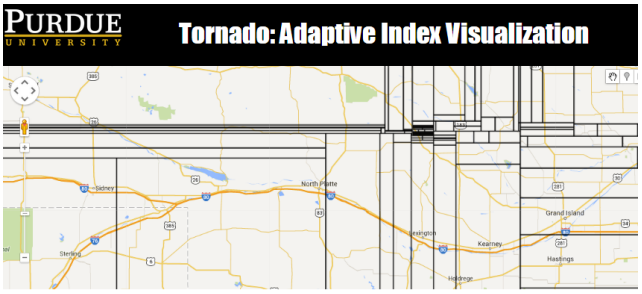


Figure 5: The index visualizer in Tornado.

```
and INSIDE(T,@currentMapView)
and INSIDE(O,@currentMapView)
and CONTAINS(O.text, "attraction")
and OVERLAPS(T.text, O.text);
```

Nearby Taxi Finder: In this scenario, a user issues a snapshot query to find nearby taxis from a specific taxi company. This query resembles a snapshot kNN query with a textual predicate and can be expressed using the following SQL statement. The output of this query visualizes the current location of the query issuer along with 5 nearest taxis that belong to the requested company.

```
RUN QUERY q2 AS
SELECT kNN FROM Vehicles AS V
WHERE CONTAINS(V.text,"Taxi","Company")
and kNN.k=5 and kNN.Focal(@myLoc);
```

In addition to the visualization of spatio-textual queries, the Tornado demo displays the internals of the adaptive index component. We visualize the effect of changing the query and data workloads on the global spatial index boundaries. To show the global index adaptivity, we give an initial partitioning of the global index. Then, we change the workload and show how the global index reacts accordingly to balance the load. Figure 5 illustrates the interface for the index adaptivity visualizer.

5. REFERENCES

[1] Hadoop. http://hadoop.apache.org/, 2015.

[2] OpenStreetMap. http://www.openstreetmap.org/, 2015.

[3] Spark. https://spark.apache.org/, 2015.

[4] Storm. https://storm.apache.org/, 2015.

[5] ZooKeeper. https://zookeeper.apache.org/, 2015.

[6] A. Eldawy and M. F. Mokbel. A demonstration of spatialhadoop: An efficient mapreduce framework for spatial data. *VLDB*, 2013.

[7] J. Lu and R. H. Guting. Parallel secondo: A practical system for large-scale processing of moving objects. In *ICDE*, 2014.

[8] Y. Ma, Y. Zhang, and X. Meng. St-hbase: a scalable data management system for massive geo-tagged objects. In *Web-Age Information Management*. Springer, 2013.

[9] A. Magdy, L. Alarabi, S. Al-Harthi, M. Musleh, T. M. Ghanem, S. Ghani, and M. F. Mokbel. Taghreed: A system for querying, analyzing, and visualizing geotagged microblogs. *SIGSPATIAL*, 2014.

[10] A. R. Mahmood, W. G. Aref, A. M. Aly, and S. Basalamah. Indexing recent trajectories of moving objects. In *SIGSPATIAL*, 2014.

[11] M. F. Mokbel, L. Alarabi, J. Bao, A. Eldawy, A. Magdy, M. Sarwat, E. Waytas, and S. Yackel. MNTG: an extensible web-based traffic generator. In *SSTD*. 2013.

[12] T. M. Sutherland, B. Liu, M. Jbantova, and E. A. Rundensteiner. D-cape: distributed and self-tuned continuous query processing. In *CIKM*, 2005.