SRC: Tornado: A Distributed Spatio-Textual Stream Processing System[*]

Ahmed R. Mahmood
Purdue University, West Lafayette, IN
amahmoo@cs.purdue.edu

1 PROBLEM AND MOTIVATION

The widespread use of GPS-enabled smart devices and the increased popularity of their applications, e.g., social networks, has led to the generation of huge amounts of spatio-textual data. This spatio-textual data needs to be processed in real-time. However, existing data management systems are either centralized or general-purpose distributed systems, e.g., Spark [2] and Storm [3]. Centralized systems do not scale and general purpose distributed systems are not optimized for the processing of spatio-textual data. In this paper, we present Tornado [7], a distributed in-memory spatio-textual stream processing system. To efficiently process spatio-textual streams, Tornado extends Storm [3] with a spatio-textual indexing layer that significantly improves the overall system performance. Tornado is adaptive, i.e., it dynamically redistributes the workload across worker processes according to changes in the distribution of spatio-textual data and queries.

2 BACKGROUND AND RELATED WORK

We live in the era of big data. However, existing data management systems are not suited for the scalable processing of spatio-textual data streams. Existing systems can be classified into the following categories [6]: (1) distributed batch-processing systems, e.g., [1, 4], that are either not optimized for spatio-textual query processing or have high processing latency, (2) distributed general-purpose streaming systems, e.g., [2, 3], that are not optimized for the processing of spatio-textual data, and (3) centralized spatio-textual streaming systems, e.g., [5], that are not scalable enough to process large amounts of spatio-textual data. Tornado is the first scalable and distributed streaming system that is able to process large amounts of spatio-textual data in real-time.

3 APPROACH AND UNIQUENESS

Tornado extends Storm [3] with spatio-textual capabilities and is designed as a pluggable module and can be seamlessly integrated with other user-defined processing. Tornado has four main layers: (1) the input layer, (2) the language layer, (3) the indexing layer, and (4) the processing layer. Figure 1 illustrates the architecture of Tornado.

3.1 The Input Layer

The input layer in Tornado is able to consume various types of spatio-textual data streams e.g., social networks posts and location-aware web searches. Moreover, Tornado can store static data, e.g., road networks and points of interests. The input layer is also responsible for consuming and parsing spatio-textual queries with the help of the language layer.

[*]Advisor: Prof. Walid G. Aref

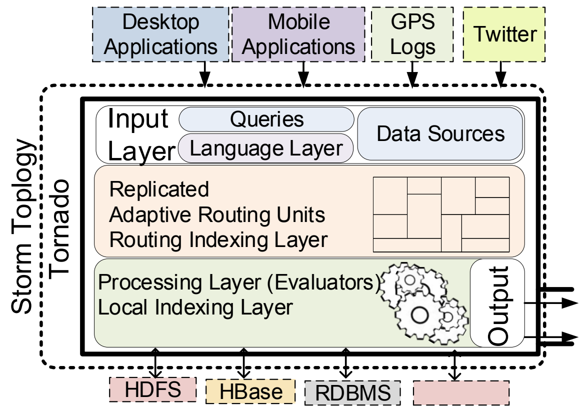


Figure 1: The layered architecture of Tornado.

3.2 The Language Layer

Recently, many complex spatio-textual queries and operators have been proposed. These proposals often address a specific spatio-textual query and use query-specific indexes and optimizations that are not applicable in other scenarios. However, a traditional data processing system, e.g., an RDBMS, uses simple relational constructs that are composable into complex queries. In contrast to using query-specific optimizations, Tornado is designed to support an SQL-like spatio-textual query language named Atlas [8]. Atlas is an extension to SQL that is able to express complex spatio-textual queries using simple constructs. Consider the following spatio-textual query that identifies groups of points of interest that are within a specific distance of each other and collectively contain a set of keywords:

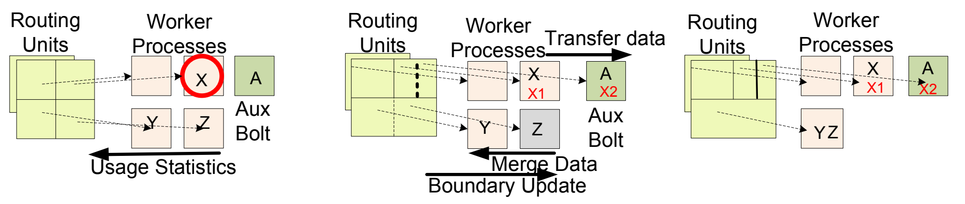
```
SELECT * FROM POIs AS p
WHERE OVERLAP("food, cinema, hotel",p.text)
PARTITION BY WITHIN_DIST(p.loc,4) AS G
ORDER BY DIAMETER (G)
HAVING CONTAINS(G.text,"food, cinema, hotel")
```

First, this query identifies points of interest that contain any of the query keywords. Then, the PARTITION BY and WITHIN_DIST clauses are used to construct groups of points of interest that are within 4 miles from each other. Finally, Atlas uses the HAVING clause to identify groups that contain all the keywords of the query.

3.3 The Indexing Layer

The indexing layer in Tornado is divided into the following sub-layers: (1) the routing layer, and (2) the local layer.

The ***routing layer*** is responsible for distributing the spatio-textual data and queries to worker processes. The routing layer uses a KD-tree-based spatio-textual index that requires minimal latency to forward data objects and queries to relevant worker processes. The



(a) Before split/merge. (b) Transient phase. (c) After split/merge.

Figure 2: Adaptivity in Tornado.

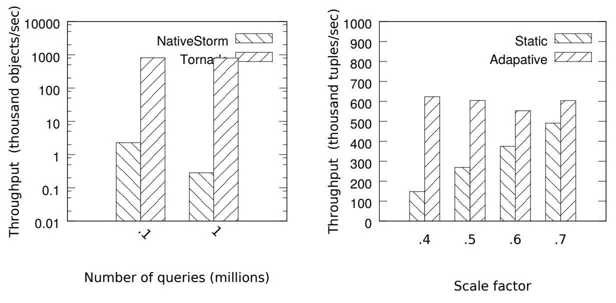
partitions of the KD-tree is overlaid on a grid structure to speed up the routing process. Also, the routing layer reduces network overhead by not sending spatio-textual data to worker processes that do not have any relevant queries. The routing layer is replicated to multiple instances to prevent performance bottlenecks. The ***local layer*** is responsible for indexing persistent spatio-textual data and continuous queries within worker processes.

The indexing layer in Tornado is adaptive and reacts to workload changes. It is challenging to ensure proper adaptivity in a distributed streaming system for the following reasons: (1) Incoming spatio-textual data can go to any instance of the replicated routing layer. This requires having a consistent structure of the routing index across all instances to guarantee the correctness of processing. (2) Workload statistics are distributed across worker processes. These statistics need to be collected periodically from worker processes with minimal network overhead. (3) Adaptivity requires migration of data and queries among worker processes that may interfere with the processing of data. The adaptivity protocol needs to ensure that there are no missing or duplicate query results at all times.

Figure 2 gives an example of load-balancing in Tornado. First, a summary of workload statistics is transmitted from worker processes to an instance of the routing layer. The routing layer identifies any overwhelmed worker processes that have workload hotspots, e.g., Process X in Figure 2(a). Also, the routing layer identifies under-utilized worker processes, e.g., Processes Y and Z. Then, Tornado goes through a transient phase to migrate part of the workload from Process X to another worker process, say A. It is typical that the underlying cluster has limited resources and Tornado cannot increase the number of worker processes indefinitely. In this case, Tornado needs to merge under-utilized processes, i.e, Processes Y and Z, to maintain a fixed number of worker processes. The transient phase is illustrated in Figure 2(b). During the transient phase, Tornado ensures that data is always forwarded to the relevant worker process for correct query evaluation. Figure 2(c) illustrates Tornado after the alleviating workload hotspots.

3.4 The Processing Layer (Evaluators)

This layer is the main working horse of Tornado and consists of a set of interconnected worker processes. Each worker process is responsible for the query evaluation over streamed data. The processing layer maintains local indexes over persistent data and continuous queries. Incoming spatio-textual data goes through the routing layer to be dispatched to relevant worker processes. Then, the incoming data is used to search local spatio-textual indexes to identify relevant spatio-textual queries.



(a) Storm Vs. Tornado. (b) Adaptivity.

Figure 3: The performance of Tornado

4 RESULTS AND CONTRIBUTION

In this section, we highlight the main performance advantages of Tornado using a real dataset of 1 Billion tweets to simulate a continuous stream of spatio-textual objects against 5 million spatio-textual queries. Figure 3(a) demonstrates that the throughput of Tornado is up to two orders of magnitude over the basic Storm streaming system. The main reason for this performance gain is that Tornado is equipped with efficient spatio-textual indexes. Figure 3(b) gives the throughput of the static and the adaptive versions of Tornado while restricting the spatial locations of data and queries to a scaled down portion of the entire space. A small scale factor condenses data and queries into a small spatial range and creates a hotspot in this range. In the static version of Tornado, the throughput degrades with the existence of hotspots. However, the adaptive version of Tornado does not suffer from poor throughput in the existence of hotspots.

REFERENCES

[1] 2017. Hadoop. http://hadoop.apache.org/. (2017).
[2] 2017. Spark. https://spark.apache.org/. (2017).
[3] 2017. Storm. https://storm.apache.org/. (2017).
[4] Youzhong Ma, Yu Zhang, and Xiaofeng Meng. 2013. ST-HBase: a scalable data management system for massive geo-tagged objects. In *Web-Age Information Management*. Springer.
[5] Amr Magdy, Louai Alarabi, Saif Al-Harthi, Mashaal Musleh, Thanaa M Ghanem, Sohaib Ghani, and Mohamed F Mokbel. 2014. Taghreed: A System for Querying, Analyzing, and Visualizing Geotagged Microblogs. *SIGSPATIAL* (2014).
[6] Ahmed Mahmood and Walid G Aref. 2017. Query Processing Techniques for Big Spatial-Keyword Data. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 1777–1782.
[7] Ahmed R Mahmood, Ahmed M Aly, Thamir Qadah, El Kindi Rezig, Anas Daghistani, Amgad Madkour, Ahmed S Abdelhamid, Mohamed S Hassan, Walid G Aref, and Saleh Basalamah. 2015. Tornado: A distributed spatio-textual stream processing system. *PVLDB* 8, 12 (2015), 2020–2023.
[8] Ahmed R Mahmood, Walid G Aref, Ahmed M Aly, and Mingjie Tang. 2016. Atlas: On the expression of spatial-keyword group queries using extended relational constructs. In *SIGSPATIAL*. ACM, 45.