SIGSPATIAL: G: Scalable Query Processing In Spatio-Textual Data Management Systems*

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, have led to the generation of huge amounts of spatio-textual data. Many applications need to process massive streams of spatio-textual data in real-time against continuous spatio-textual queries. For example, in location-aware ad targeting publish/subscribe systems, it is required to disseminate millions of ads and promotions to millions of users based on the locations and textual profiles of users. These applications resulted in the proliferation of several complex spatio-textual queries, e.g., spatio-textual group queries that search for groups of spatio-textual objects that are very close to each other and collectively cover a set of keywords.

Almost all online textual transactions, e.g., web searches, have an associated spatial trace that can be inferred from the GPS coordinates or the IP address of the user. Many industrial companies that provide online services offload their processing to cloud providers. Cloud providers charge industrial companies for hosting their applications. The higher the computational and the storage requirements of hosting an application, the higher the monetary cost charged. Hence, the optimization of spatio-textual stream processing has a significant impact when the solution is able to reduce the financial cost incurred for hosting spatio-textual applications.

Existing data management systems are either centralized, e.g., AP-tree [21] 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. These systems do not account for some important properties of spatio-textual data, e.g., the non-uniform and varying spatial and textual distributions of streamed data and queries. Also, existing systems use query-specific indexes and algorithms to answer complex spatio-textual queries. However, traditional data management systems. e.g., relational database management systems (RDBMSs) do not use query specific indexes and algorithms. These systems use building blocks and operators, e.g., SELECT, PROJECT, and JOINs that are composable to express complex queries.

In this paper, we present Tornado [13–16], a distributed in-memory spatio-textual stream processing system. To efficiently process spatio-textual streams, Tornado extends Storm [3] with a two-layered spatio-textual indexing layer that significantly improves the overall system performance. We design and implement spatio-textual indexes that account for the properties of spatio-textual data and queries to be used in the two-layered spatio-textual indexing layer. 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. Tornado achieves around an order of magnitude throughput improvement over the baseline approach. In a step towards systematic spatio-textual query processing, we introduce spatio-textual building blocks that are able to express a wide range of complex spatio-textual queries.

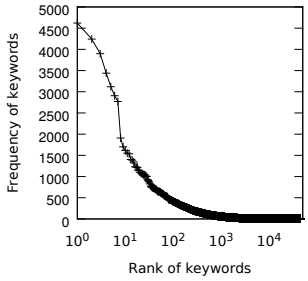


Figure 1: Zipfian distribution of query keywords.

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 [12]: (1) distributed batch-processing systems, e.g., [1, 9], 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., [10], 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.

Spatio-textual data often have non-uniform spatial and textual distributions. Figure 1 illustrates that the frequencies of Twitter keywords follow a Zipfian distribution. Also, popular and frequent keywords are not the same for different spatial regions. Recently, several spatio-textual indexes [6] have been introduced. These indexes often integrate a spatial index e.g., the R-tree [7] or the spatial Grid [18], with a textual index e.g., the inverted list [22] or the ordered keyword trie [8]. However, existing spatio-textual indexes do account for the underlying distribution of spatio-textual data.

There exists a wide range of spatio-textual queries. However, few spatio-textual query languages have been proposed. There exist

*Advisor: Prof. Walid G. Aref

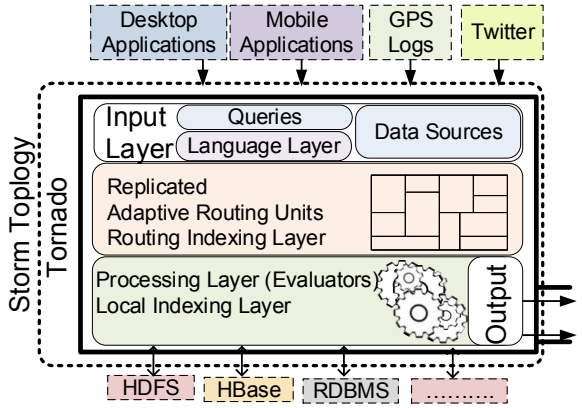


Figure 2: The layered architecture of Tornado.

several spatial-only [5, 17] and textual-only [17, 20] query languages. These query languages cannot support complex spatio-textual queries. The only proposal for a spatio-textual query language is the microblogs query language MQL [11] that is also not able to express complex spatio-textual queries.

3 APPROACH AND UNIQUENESS

In this section, we present the structure of Tornado and its main features. 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 indexing layer, (3) the processing layer, and (4) the language layer. Figure 2 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.

3.2 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. Distributed streaming systems, e.g., Storm, provide an infrastructure of workload distribution to different worker processes. Worker processes in a general purpose streaming system can be categorized into source processes and execution processes. Source processes inject streamed data and queries into the distributed system. Execution processes perform the user-defined logic. Figure 3(a) illustrates a basic approach for processing continuous spatio-textual queries in a general distributed streaming system. This approach is not spatio-textual aware and does not assume any knowledge about the properties streamed data and queries. In this basic approach, data are partitioned to multiple execution processes and queries are replicated to all execution processes to ensure that all queries are evaluated properly. However, this basic approach has limited scalability and incurs high computational and storage overhead due to query replication. In this approach, every execution process maintains all queries and evaluates all queries against streamed data.

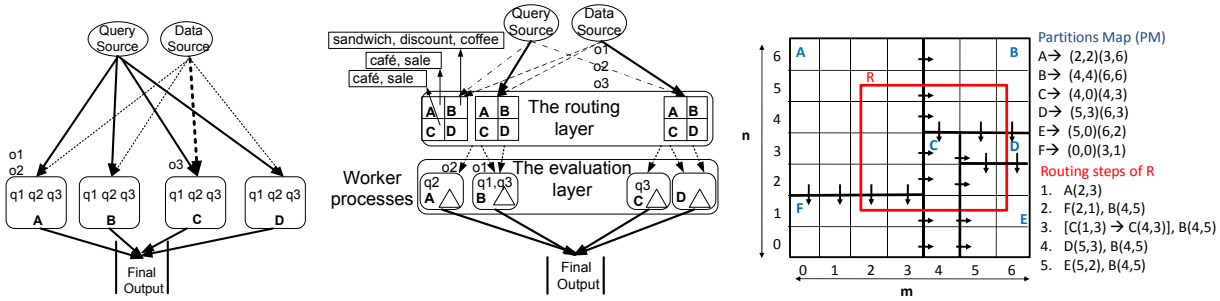
Figure 3(b) illustrates the spatio-textual-aware workload partitioning that is adopted in Tornado. The routing layer is introduced to partition both streamed data and queries to execution processes, i.e., evaluators. The routing layer avoids replicating queries by using the spatial and textual properties of data and queries to ensure that relevant data and queries are collocated in the same execution processes, i.e., evaluator.

The routing layer uses a Grid-based spatio-textual index termed the augmented grid (A-Grid), that requires minimal latency to forward data objects and queries to relevant worker processes. The A-Grid partitions the space into non-overlapping partitions. These partitions are calculated using KD-tree-based [19] partitioning of a training workload. Then, these partitions are overlaid on the grid cells of the A-Grid. Then, A-Grid cells get *augmented* with pointers to help quickly identify neighboring partitions using a *neighbor-based* routing algorithm. The structure of the A-Grid is illustrated in Figure 3(c).

The routing layer reduces network overhead by not forwarding spatio-textual data to evaluators that do not have any relevant queries. As illustrated in Figure 3(b), the routing layer maintains a textual summary of all queries in evaluators. This textual summary is checked against streamed data prior to routing to evaluators. The routing layer is replicated to multiple instances to prevent performance bottlenecks.

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 4 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 4(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 4(b). During the transient phase, Tornado ensures that data is always forwarded to the relevant worker process for correct query evaluation. Figure 4(c) illustrates Tornado after the alleviating workload hotspots.

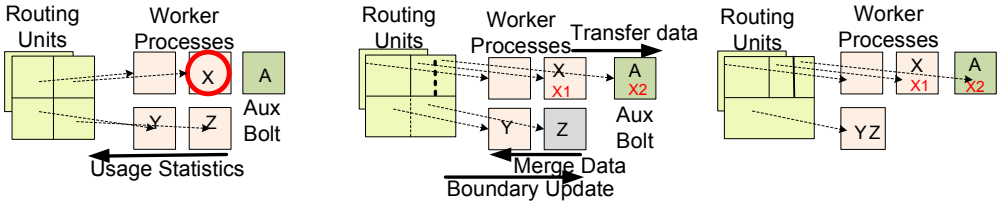


(a) Non spatio-textual aware query evaluation

(b) The routing units and evaluators.

(c) Neighbor-based query routing in Tornado.

Figure 3: Data partitioning approaches.



(a) Before split/merge.

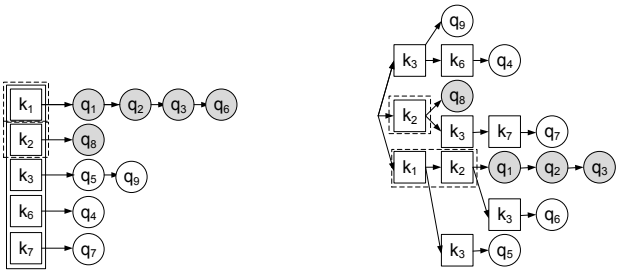
(b) Transient phase.

(c) After split/merge.

Figure 4: The split/merge operation.

The ***local layer*** is responsible for indexing persistent spatio-textual data and continuous queries within worker processes. In the local index layer, we introduce an efficient spatio-textual index termed FAST [13]. FAST integrates a textual index termed the *Adaptive Keyword Index AKI* [13] with the spatial pyramid structure [4]. AKI is an adaptive textual index that avoids the limitations of other widely adopted textual indexes, i.e., the *ranked inverted list (RIL) [22]* and the *ordered keyword trie (OKT) [8]*. RIL is based on the hashmap structure, where indexed objects get attached to a posting lists of keywords as illustrated in Figure 5(a). OKT is similar to the traditional trie structure that indexes string using single characters. However, OKT stores keywords in its nodes rather than single character as illustrated in Figure 5(b). RIL has low memory requirements and tends to have poor search performance for keywords that have long posting list. OKT has good search performance with high memory overhead ALso, OKT requires knowing the entire vocabulary of indexed objects. AKI avoids these limitations by initially indexing textual objects as a RIL as illustrated in Figure 6(a). When the number of textual objects attached to a single keyword exceeds a specific threshold termed the *frequent-keyword threshold* θ, AKI starts indexing objects using more keywords to improve its textual discrimination power as illustrated in Figure 6(b).

AKI is a text index and does not account for the spatial attribute of spatio-textual data. FAST integrates the spatial pyramid with AKI to introduce spatial discrimination when textual discrimination is lost, i.e., when all indexed objects have the same keywords at different locations. The spatial pyramid is a multi-level grid index, where lower pyramid levels have finer granularities. Figure 7 illustrates the structure of FAST. When many items have the same sets of keywords, the items descend to a lower pyramid level to take advantage of the improved spatial discrimination of the lower pyramid level. FAST is extremely memory efficient and it avoids replicating indexed spatio-textual items by sharing textual structures among the spatial pyramid cells.



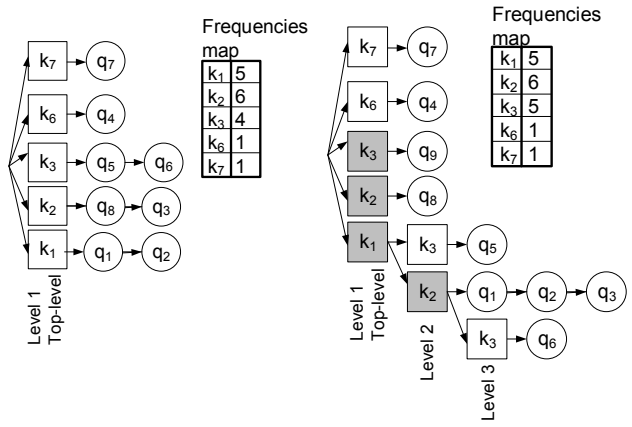
(a) Ranked-keyword inverted list (RIL)

(b) Ordered-keyword trie (OKT)

Figure 5: Relevant textual indexes and spatio-textual indexes.

3.3 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. FAST is used inside processing layer to improve the overall scalability of Tornado. 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) Before inserting q_9 (b) After inserting q_9

Figure 6: The adaptive keyword index *AKI*.

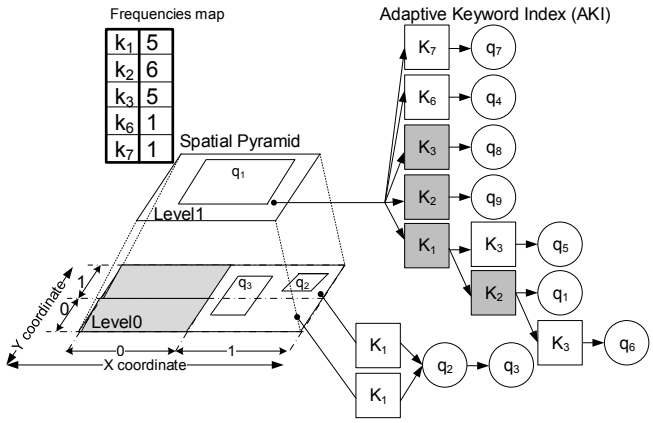


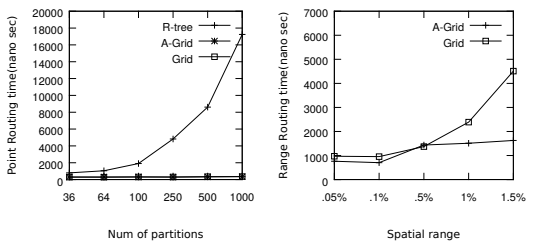
Figure 7: The structure of FAST.

3.4 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 [15]. Atlas is a specification of an extension to SQL that is able to express complex spatio-textual queries using simple constructs. The extended syntax of the SELECT statement is as follows:

```
SELECT {*|attr1 [AS alias][,attr2,...]}
FROM source_name1 [,source_name2,...]
[WHERE condition]
[ORDER BY F(arg_list)]
[LIMIT {k|condition}]

SELECT grp_attr1 [AS alias][,grp_attr2,...],
AGGR_F [AS alias](attr_list)
FROM source_name1 [,source_name2,...]
[WHERE condition]
{PARTITION BY} grp_attr_list AS group_alias
[ORDER BY F(grp_arg_list)]
[LIMIT k]
[HAVING {condition}]
```



(a) Point routing time (b) Range routing time

Figure 8: Spatial routing time for points and ranges.

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. The full realization of the operators of Atlas in Tornado is a future work.

4 RESULTS AND CONTRIBUTION

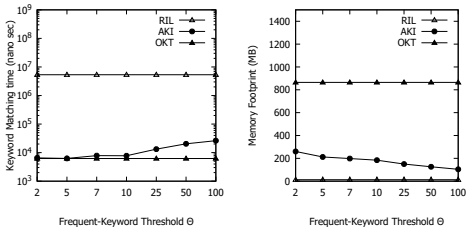
In this section, we highlight the main performance advantages of Tornado and its internal components, i.e., A-Grid, AKI, and FAST using a real dataset of 1 Billion tweets to simulate a continuous stream of spatio-textual objects against millions of spatio-textual queries.

4.1 The performance of A-Grid

Figure 8(a) illustrates that the point routing time in the A-Grid is much better than the point routing time in the R-tree and is equivalent to the point routing time in the traditional Grid. Also, Figure 8(b) shows that range routing time in the A-Grid is better than the range routing time in the traditional grid. Hence, the A-Grid achieves efficient routing performance when routing point and range data. This improves overall performance of the routing layer in Tornado.

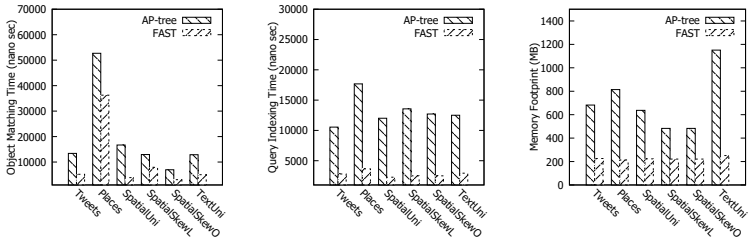
4.2 The performance of AKI

Figure 9 illustrates the performance of AKI against RIL and OKT while varying the frequent-keyword threshold θ. Figure 9(a) demonstrates that the searching performance of AKI is very comparable to



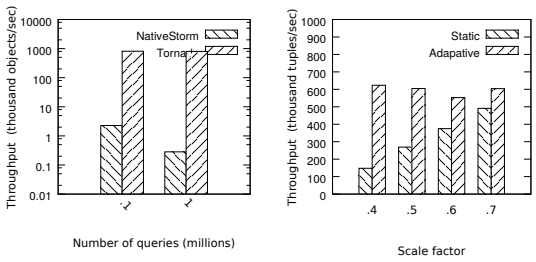
(a) Searching time AKI (b) Memory footprint AKI

Figure 9: *Frequent-keyword* threshold θ



(a) Searching time (b) Indexing time (c) Memory footprint

Figure 10: Performance under different datasets



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

Figure 11: The performance of Tornado

the searching performance of OKT and is much better than that of RIL. Figure 9(b) shows that the memory footprint of AKI is much less than that of OKT and very comparable to the memory footprint of RIL. Hence, AKI achieves efficient searching performance while requiring low memory footprint.

4.3 The performance of FAST

Figure 10 illustrates the performance of FAST against the state-of-art index, i.e., the AP-tree [21], using multiple real and synthetic datasets. Figure 10(a) shows that FAST is up to 3x faster than the AP-tree in terms of the searching performance. Figure 10(b) demonstrates that FAST is up to 5x faster than the AP-tree in terms of the indexing performance. Figure 10(c) shows that FAST requires up to one third of the memory required by the AP-tree. The efficiency of FAST in the indexing and the searching operations when combined with its low memory requirements significantly improve the scalability of Tornado.

4.4 The overall performance of Tornado

Figure 11(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 11(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] 2018. Hadoop. http://hadoop.apache.org/. (2018).
[2] 2018. Spark. https://spark.apache.org/. (2018).
[3] 2018. Storm. https://storm.apache.org/. (2018).
[4] Walid G Aref and Hanan Samet. 1990. Efficient processing of window queries in the pyramid data structure. In *Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. ACM, 265–272.
[5] Walid G Aref and Hanan Samet. 1991. Extending a DBMS with spatial operations. In *Advances in Spatial Databases*. 297–318.
[6] Lisi Chen, Gao Cong, Christian S Jensen, and Dingming Wu. 2013. Spatial keyword query processing: an experimental evaluation. *VLDB* (2013).
[7] Antonin Guttman. 1984. *R-trees: a dynamic index structure for spatial searching*. Vol. 14. ACM.
[8] Zeinab Hmedeh, Harris Kourdounakis, Vassilis Christophides, Cédric Du Mouza, Michel Scholl, and Nicolas Travers. 2012. Subscription indexes for web syndication systems. In *Proceedings of the 15th International Conference on Extending Database Technology*. ACM, 312–323.
[9] 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.
[10] 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. In *SIGSPATIAL*. 163–172.
[11] Amr Magdy and Mohamed F Mokbel. 2015. Towards a Microblogs Data Management System. In *MDM*, Vol. 1. 271–278.
[12] 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.
[13] Ahmed R Mahmood, Ahmed M Aly, and Walid G Aref. 2018. FAST: Frequency-Aware Indexing for Spatio-Textual Data Streams. (2018).
[14] 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.
[15] 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.
[16] Ahmed R Mahmood, Anas Daghistani, Ahmed M Aly, Walid G Aref, Mingjie Tang, Saleh Basalamah, and Sunil Prabhakar. 2017. Adaptive Processing of Spatial-Keyword Data Over a Distributed Streaming Cluster. *arXiv preprint arXiv:1709.02533* (2017).
[17] Jim Melton and Andrew Eisenberg. 2001. SQL multimedia and application packages (SQL/MM). *Sigmod Record* 30, 4 (2001), 97–102.
[18] Jürg Nievergelt, Hans Hinterberger, and Kenneth C Sevcik. 1984. The grid file: An adaptable, symmetric multikey file structure. *TODS* 9, 1 (1984), 38–71.
[19] Beng Chin Ooi, Ken J McDonell, and Ron Sacks-Davis. 1987. Spatial kd-tree: An indexing mechanism for spatial databases. In *IEEE COMPSAC*, Vol. 87. 85.
[20] Jaehui Park and Sang-Goo Lee. 2011. Keyword search in relational databases. *Knowledge and Information Systems* 26, 2 (2011), 175–193.
[21] Xiang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Wei Wang. 2015. Ap-tree: Efficiently support continuous spatial-keyword queries over stream. In *ICDE*. 1107–1118.
[22] Justin Zobel and Alistair Moffat. 2006. Inverted files for text search engines. *ACM computing surveys* 38, 2 (2006), 6.