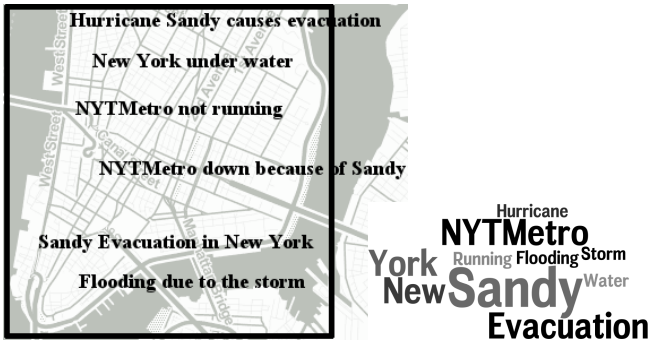
Scalable Top-k Spatio-Temporal Term Querying

Anders Skovsgaard[#1], Darius Sidlauskas[#2], Christian S. Jensen[*]

[#]*Department of Computer Science*
Aarhus University, Denmark
{[1]anderssk, [2]dariuss}@cs.au.dk

[*]*Department of Computer Science*
Aalborg University, Denmark
csj@cs.aau.dk

***Abstract*—With the rapidly increasing deployment of Internet-connected, location-aware mobile devices, very large and increasing amounts of geo-tagged and timestamped user-generated content, such as microblog posts, are being generated. We present indexing, update, and query processing techniques that are capable of providing the top-k terms seen in posts in a user-specified spatio-temporal range. The techniques enable interactive response times in the millisecond range in a realistic setting where the arrival rate of posts exceeds today's average tweet arrival rate by a factor of 4–10. The techniques adaptively maintain the most frequent items at various spatial and temporal granularities. They extend existing frequent item counting techniques to maintain exact counts rather than approximations. An extensive empirical study with a large collection of geo-tagged tweets shows that the proposed techniques enable online aggregation and query processing at scale in realistic settings.**



(a) Messages (b) Tag Cloud

Fig. 1: Example of Messages in New York, USA.

I. Introduction

The digital universe is expanding exponentially, and the amount of text content available on the web, including news items, web pages, and microblog posts, grows rapidly. Today, everyone can generate such content. In particular, services such as Twitter, Facebook, and Blogger make it easy for all to contribute.

Users of social media services often generate similar content in response to events that catch their attention. For example, when natural disasters occur, multiple users are likely to report on this independently. Some users may discuss evacuations and the traffic situation, while other users may simultaneously discuss unrelated topics such as food recipes or a sporting event. With thousands of pieces of content being made available each second, techniques are needed to maintain an overview of what occupies minds of users.

Proposals existing that are capable of finding currently popular topics in streaming text content [24], [26]. We extend these by supporting also queries on past data and by allowing spatio-temporal range queries. By supporting queries that retrieve the top-k most frequent terms in content in user-specified spatio-temporal regions, we support different services that aim to find the "word in the street" or "talk of the town" in some time period. To illustrate, Figure 1a shows a few tweets located in a query region that covers part of New York City during some time period. The most popular terms can be represented by a tag-cloud, as shown in Figure 1b.

In a real-world setting, spatio-temporal regions may contain millions of tweets.

A key design goal for the techniques needed to support this functionality is scalability. Specifically, we seek a solution that is capable of supporting the entire world and a long history, as well as supporting a stream with many more posts per second than what the average of some 3,000 tweets per second that Twitter sees (and of which only a fraction is geo-tagged). This goal is set to achieve a future-proof solution that anticipates the rapid growth in geo-tagged posts.

Index structures, e.g., based on the R-tree, have been proposed to support spatio-temporal queries. However, these are not well suited for our setting with a rapid content stream and spatio-temporal aggregate queries [32]. We thus propose new indexing techniques that aim to support efficient querying of streaming and historical data. We extend a popular algorithm that maintains static-sized summaries of the most frequent items in data streams to support dynamic, variable-sized summaries so it can adapt to diverse data. Further, we provide a new merging algorithm for these summaries.

The paper makes four contributions. First, we provide techniques capable of supporting a holistic aggregate function [12]: the top-k most frequent terms. We propose a variant of frequent item counting that maintains top-k most frequent terms error-free. This allows us to combine aggregates as needed to support arbitrary spatio-temporal regions with low probability of introducing errors, and it allows us to provide correctness guarantees.

Second, we provide techniques that maintain and use spatio-temporal aggregates that grow and shrink according to the activity (the number of distinct items to aggregate) and the required k. The aggregates are partially persistent, i.e., all versions can be accessed, but only the current version can be modified. Once aggregated, the top-k terms are compact in comparison to the currently active aggregates.

Third, we integrate the above techniques into the Adaptive Frequent Item Aggregator (AFIA) system that supports spatio-temporal top-k frequent term queries. The finest spatial granularity supported is 100×100 m^2, as finer granularities make little sense in our setting, and the finest temporal granularity is an hour.

Fourth, we study AFIA empirically using a real-world data stream from Twitter. On server class machines, AFIA supports a stream with more than 13,000 tweets per second. One month of aggregates requires up to 120 GB of memory. The accuracy depends on the number of precomputed aggregates needed to answer a query. Since we maintain aggregates at multiple spatial and temporal granularities, the lowest observed accuracy was 97%. The study suggests that AFIA meets the scalability design goal.

The remainder of the paper is structured as follows. Section II-A defines the problem and covers related work. Section III presents the indexing and aggregation framework. An experimental evaluation is given in Section IV. Finally, we conclude in Section V.

II. Preliminaries and Related Work

A. Problem Definition

Let D be a set of spatio-temporal web objects $o = (\lambda, doc, ts)$, where λ is a point location in 2D Euclidean space, doc is a text document, and ts is a time point. The score of a term t for a set S of objects, $score(t, S)$, is the count of objects in S whose text document contains t:

$$score(t, S) = |\{o \in S \mid t \in o.doc\}| \quad (1)$$

A top-k most popular terms query $q = (R, k, I)$ returns a pair consisting of (i) k terms from objects in D that belong to the spatio-temporal range defined by 2D rectangle R and time interval $I = [t_s, t_e]$ and (ii) an integer $k_g \leq k$. The first k_g terms are guaranteed to have the highest scores for the set considered, and the remaining $k - k_g$ terms are approximate. When $k_g = k$, the result is exact.

The query is designed to take advantage of the setting, where exact results are not necessary, in order to achieve high performance. We aim to compute results were k_g is close to k and where the $k - k_g$ unguaranteed terms either are top-k terms or are close to being so.

Example 1. Consider six objects with these documents:
s_1 = "Hurricane Sandy causes evacuation of NYTMetro."
s_2 = "NYC under water."
s_3 = "NYTMetro not running."
s_4 = "NYTMetro down because of sandy."
s_5 = "Sandy Evacuation in New York."
s_6 = "Flooding due to the storm."
Ignoring word case and so-called stop words (in grey color), 17 terms are considered in total. An exact query with $k = 3$ on these objects yields the result $(\{\text{sandy}, \text{nytmetro}, \text{evacuation}\}, 3)$. □

B. Aggregation

Aggregation is an important kind of operation and has been studied widely. In OLAP settings, it is typical to precompute intermediate aggregation results and then to reuse these for computing higher level results. The choice of which intermediate results to precompute is based on frequently-asked queries and apriori known data groupings and hierarchies [5], [12], [14]. In stream processing, running aggregates of stream data may be maintained so that up-to-date summaries of the data are readily available (e.g., [22]).

In a spatio-temporal setting, Papadias et al. [30] propose the aggregation R-tree (aR-tree) that augments each node with an aggregate value (e.g., count) for all objects in the node's subtree. Then, during querying, there is no need to perform the aggregation for objects in MBRs that are covered by the query region. Mamoulis et al. [21] show how using the aR-tree to support top-k queries. Yang and Widom [36] propose the SB-tree that maintains a hierarchy of intervals associated with partially computed aggregates.

There are three types of aggregation functions [12]: distributive, algebraic and holistic. Distributive (e.g., count, sum, max) and algebraic (e.g., avg) aggregates can be computed by further aggregating intermediate aggregates, while holistic aggregates cannot. The pre-aggregation techniques described above work only for non-holistic aggregation functions, while the top-k query considered in this paper is holistic. We are not aware of any existing spatio-temporal grouping techniques that support holistic aggregates.

C. Frequent Item Counting

The aggregation of frequent items in data streams has also been studied widely. The ϕ-frequent items in a set S are $\{i \in S \mid f_i > \phi|S|\}$, where f_i is the frequency of an item i in S. In Example 1, $f_{\text{sandy}} = 3$, $f_{\text{nytmetro}} = 3$, $f_{\text{evacuation}} = 2$, $f_{\text{hurricane}} = 1$, etc. For $\phi = 0.1$, the frequent items are $\{\text{sandy}, \text{nytmetro}, \text{evacuation}\}$. Since solving the ϕ-frequent problem requires linear space ($\Omega(|S|)$), an approximate version is often considered: the ϵ-approximate frequent items in a set S are $\{i \in S \mid f_i > (\phi - \epsilon)|S|\}$.

Approaches to frequent item counting can be categorized as counter-based and sketch-based. The former monitor the items in a stream using a fixed number of counters. If an item monitored by a counter arrives, its count is incremented. Otherwise, depending on the algorithm, the item is ignored, or it takes the place of a currently monitored item. Notable counter-based algorithms include LossyCounting [23], Frequent [11], [16], [29], and SpaceSaving [28].

Sketch-based approaches maintain the approximate frequency for all items using hashing. Items are mapped into a (smaller) space of counters, and a hashed-to counter is updated for every occurrence of a corresponding item. Notable sketch-based algorithms include CountSketch [6], GroupTest [10], and Count-Min Sketch [9].

Because the sketch-based approaches provide no guarantees about the relative order of items or their estimated frequency and also suffer from much higher per item processing costs, we turn to counter-based techniques. In particular, we adapt the SpaceSaving algorithm. In addition to finding

a set of frequent items, this algorithm also finds the top-k items [28]. It can also provide relative order guarantees for individual items because each counter value is associated with its maximum error. Further, our micro-benchmarks as well as recent experimental evaluations [8], [22] find that SpaceSaving is a top performer.

SpaceSaving is outlined in Algorithm 1. Each monitored item i is associated with its count, c_i, and a maximum error, Δ_i. The counters are initialized by the first m distinct items,

Algorithm 1: `SpaceSaving`(counters m, stream S)

```
1  T ← ∅ ;                                          // monitored items
2  foreach i ∈ S do
3      if i ∈ T then
4          c_i ← c_i + 1 ;
5      else if |T| < m then
6          T ← T ∪ {i} ;
7          c_i ← 1 ;
8      else
9          j ← min_{j∈T} c_j ;                      // item with min count
10         c_i ← c_j + 1 ;                          // new count
11         Δ_i ← c_j ;                              // max error
12         T ← T ∪ {i} \ {j} ;
```

and their exact counts (Lines 3–8). When an un-monitored item arrives, it replaces a currently monitored item with the lowest count. As Lines 10–13 indicate, the new item gets the maximum possible count and records its maximum error in Δ_i. Intuitively, frequent items reside in counters with large values and will not be hit by infrequent items that grow more slowly. SpaceSaving is guaranteed to capture all ϵ-frequent items with $m = 1/\epsilon$ number of counters and the maximum counter error is $\lfloor n/m \rfloor$ [28].

SpaceSaving can provide three levels of guarantees. First, it can report only items that are guaranteed to be ϵ-approximate frequent: report a monitored item i if $c_i - \Delta_i > \phi n$. Second, it guarantees that the first k items are the top-k most frequent items if $\forall_{i \leq k}\ (c_i - \Delta_i \geq c_{k+1})$. Third, it guarantees that the top-k items are correctly ordered if $\forall_{i \leq k}\ (c_i - \Delta_i \geq c_{i+1})$.

We extend SpaceSaving to support exact top-k ordered queries. To support arbitrary spatio-temporal regions, we rely on the ability to merge aggregated top-k terms across multiple spatio-temporal granularities. In doing so, we rely on the notion of *mergeability* of data summaries introduced by Agarwal et al. [1]. They show that frequent item aggregates computed using Frequent (or SpaceSaving) are mergeable and preserve space and error guarantees. Our SpaceSaving extension and its accompanying merging algorithm preserve exact query results. We achieve this by trading the space (memory) for being error-free in an adaptive manner.

D. Related Systems

Several recent systems target social media streams. BlogScope [4] (and its commercial counterpart, Sysomos) collects text documents from news feeds, mailing lists, forums, newsgroups, blogs, etc. Each document is temporally ordered and is associated with the profile of its author, which may include a location. The system supports the discovery and tracking of real-world entities (stories, events, etc.) [3], the monitoring of most popular keywords ("trends") [27], and the monitoring of temporal and/or spatial bursts [25]. However, the system does not aggregate keywords according to user-specified spatio-temporal regions. Also, only currently popular keywords are identified (with a span of up to few minutes), whereas our system supports also queries on the past.

NewsStand [34], a spatio-textual news aggregator, extracts geographic content from RSS feeds and groups articles into story clusters. Users can then retrieve stories relevant to query keywords and geographic regions. TwitterStand [31] uses tweets instead of RSS feeds. Both systems employ a spatio-textual search engine [20] that was recently updated to support spatio-temporal querying [17] on a small ProMED dataset. News aggregation involves significantly more computation and storage than does our problem. However, the systems also support much lower updates rates. NewsStand processes up to 50,000 RSS feeds per day [19], while TwitterStand monitors tweets generated by 2,000 handpicked users who publish news. Both applications employ transactional database technology (PostgreSQL) for backend processing. In contrast AFIA, aims to examine all tweets and to provide exact counts for most popular keywords in user-specified spatio-temporal regions in interactive time. This calls for the ability to process up to tens of thousands tweets per second. Therefore, AFIA is a stand-alone system that employs spatio-temporal indexing.

Recently several recommender systems based on user-generated posts have been proposed [15], [33], [35]. Limosa [35] provides user recommendations based on their geographic interest by exploiting explicit geotags associated with tweets and by extracting locations from the tweet itself. As only 0.42% of tweets are geotagged [7], this is very relevant; however, it is orthogonal to our work.

III. Proposed Solution

We present a framework that maintains spatio-temporal grids at multiple granularities that partition Earth and time. A precomputed summary is maintained for each grid cell. The sizes of the summaries adapt dynamically to the data, and a summary merging algorithm enables query processing.

A. Adaptive Frequent Item Aggregator

We proceed to describe the multi-granularity index structure and then present an overview of the framework.

1) Grid-Based Indexing of Precomputed Summaries: Large volumes of queries combined with large spatio-temporal query regions can lead to the need for aggregating billions of messages. To achieve interactive response times, we introduce an index structure that supports pre-aggregation at multiple granularities. This way, a user-specified spatio-temporal range query can be partitioned into a collection of coarsest spatio-temporal regions for which aggregates are available, and these aggregates can then be combined efficiently to produce the query result.

We employ a static grid-based approach that uses uniform grids [2] with predefined and fixed cell sizes. By introducing

multiple layers of grids with increasing cell sizes, we partition space at multiple granularities. By precomputing information about the most frequent terms in each grid cell, queries with regions of any size larger than the finest granularity cell can be efficiently supported. By indexing the information precomputed for each cell in a hash table, this approach supports constant-time cell lookup.

Therefore, to efficiently support queries with different spatial extents, we divide the world into a grid with multiple granularities as seen in Figure 2. The lowest granularity defines

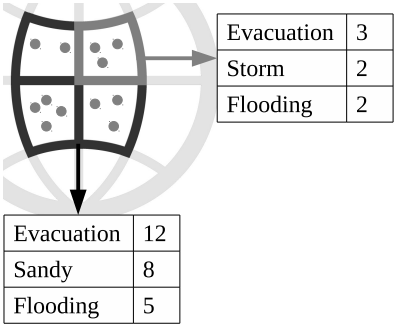


Fig. 2: Multiple Spatial Granularities

the spatial accuracy of our summaries (which translates to GPS accuracy, for instance). Precomputed summaries are associated with each cell at each granularity. Coarse granularities are used to more efficiently support queries with large regions as they obviate the need for many summaries at finer granularities.

2) Frequent Term Counting: Each lowest-granularity cell covers a small part of the world, and different cells may receive very different amounts of objects. Also, the terms in objects may vary substantially, e.g., because users use different languages. Thus, any given cell may see a large number of terms that may not be used in other cells. To store, for each cell, each term in list, e.g., ordered by frequency, requires linear space and is infeasible for large-scale processing.

Instead, we keep track only of the most frequent terms in each cell. This is achieved by extending the counter-based *SpaceSaving* algorithm described in Section III-B2. As a result, only a fraction of the terms is stored, while we maintain guarantees about the results of queries. The precomputed most frequent term counts are maintained at every object insertion. The result is a ranked summary for each cell at each granularity of the terms used in the objects that fall into the cell, as shown in Figure 2.

3) Temporal Support: To accommodate the temporal extent of queries, the spatial grid is extended with a temporal dimension. The finest temporal granularity is the smallest query-enabled time interval, e.g., an hour. We create a new instance of every spatial grid cell for every new such time interval. This is illustrated in Figure 3, where each cell has frequent term counts for each time interval. This arrangement enables efficient support for temporal query ranges at the finest granularity, but it does not offer support for long temporal query ranges, e.g., weeks or months.

To support queries with arbitrary temporal extent, we also create new grid cells for multiple time granularities. For

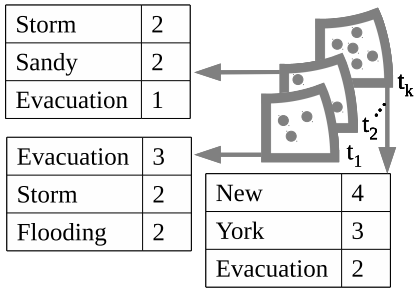


Fig. 3: Temporal Dimension

example we may define month, week, day, and hour as our time granularities. For each spatial cell the corresponding objects are grouped for each time granule. The temporal granules each maintain a summary of frequent term counts. This is illustrated in Figure 4. With this arrangement, a two-week query result

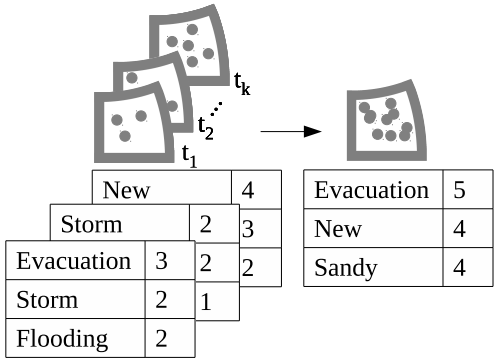


Fig. 4: Multiple Temporal Granularities

can be obtained by merging only two frequent term summaries.

4) Combining Multiple Granularities: For each granule, we store aggregate information that allows us to return the top-k result for the granule. Since a coarser granule aggregates the same terms as a number of finer granules at a finer granularity, the coarser granule can be computed by merging the corresponding finer granules [1]. However, the computed aggregate looses its accuracy. Figure 5 illustrates the problem. A coarse granule contains two finer granules, and we maintain information needed to support $k = 3$ for these two granules. Combining them yields an incorrect top-3 list for the coarse granule. The correct result contains "Sandy" with count 4.

To avoid such incorrect merging, we maintain separate frequent term summaries for every granularity. Thus, when an object is inserted, its terms are reflected in the aggregate information stored for each granule the object falls into. The same principle applies to the temporal dimension.

5) System Overview: Figure 6 shows an overview of the adaptive frequent item aggregator (AFIA) system. The index structure in the middle consists of the above-mentioned spatial (SI) and temporal (TI) granularities aggregated with frequent item counting summaries (FIC).

High-level pseudo-code for stream data processing in AFIA is given in Algorithm 2. Each message from the data stream

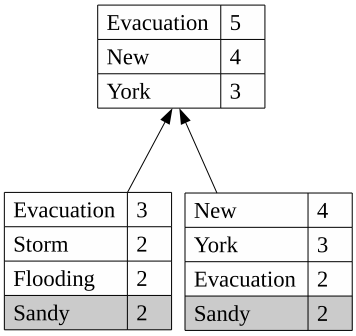


Fig. 5: Example of Incorrect Merging

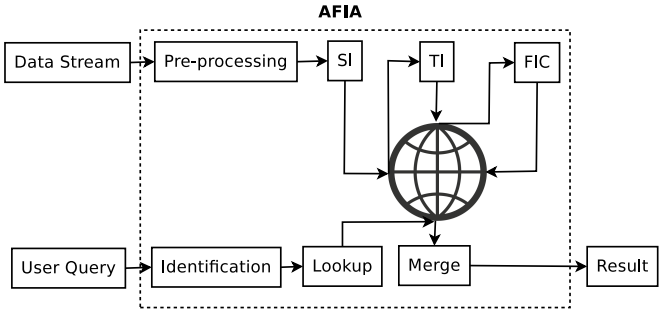


Fig. 6: The AFIA System

is pre-processed by splitting the message into terms and removing stop words. The corresponding summary is found in Line 3. As seen from Figure 6, finding the summary entails lookups in the index structure. If a summary is able to hold the new message, it is inserted as shown in Line 4. However, in some cases, it may be necessary to expand a summary before insertion. This is described in detail in Section III-B2. If the summary cannot be expanded, it becomes inactive, and a new empty summary is created. This is called a *checkpoint*, which is covered in Section III-B3.

Algorithm 2: `AFIAStreamProcessing`(stream S)

```
1 foreach i ∈ S do
2     preprocess(i) ;
3     summary ← find(i) ;            // find the summary
4     if ! summary.insert(i) then
5         checkpoint(summary) ;
```

High-level pseudo-code for query processing is given in Algorithm 3. First, the query region is mapped to the corresponding cells. Next, a lookup is performed for each cell that each produces a set of summaries as seen in Lines 3–4. Each summary is a partial result. The lookup utilizes the query parameters $q.R$ and $q.I$ to limit the number of partial results. Finally, the results are merged to compute the top-k terms and k_g (detailed in Section III-C).

B. Dynamic Summaries

1) Motivation: Existing frequent item counting techniques utilize static-sized summaries. While this works when the vocabulary is known beforehand, any static summary size falls short when the vocabulary changes considerably. And since we aim to cover the entire world at multiple spatial and temporal granularities, the vocabulary is guaranteed to vary across cells. If we use static-sized summaries with too few counters, we cannot use the summaries to determine the top-k most frequent terms. In Example 2, too few counters are used to capture the top-2 most frequent terms. An obvious solution is to use summaries with an extremely large number of counts. However, this is either not feasible or unattractive because it require excessive storage space. Instead, we develop a new technique that dynamically adapts the number of counters in a summary to accommodate the changes in the data.

Algorithm 3: [R, k_g] `AFIAQuery`(query Q)

```
1 cells ← retrieve cells that Q.R covers ;
2 summaries ← empty list of summaries ;
3 foreach c ∈ cells do
4     summaries.add(lookup(c, Q)) ;
5 [R, k_g] ← merge(Q.k, summaries) ;
6 return [R, k_g] ;
```

Example 2. Assume we wish to support top-2 queries with the SpaceSaving algorithm and that a summary with 2 counters is used. The following terms are inserted in the given order: $evacuation$, $evacuation$, $sandy$, $evacuation$, $sandy$, $storm$.

The summary then becomes: $\{evacuation = 3(\Delta = 0), storm = 3(\Delta = 2)\}$ (where Δ captures the maximum over-counting error. In contrast, the correct result is $\{evacuation, sandy\}$. □

2) Aggressive Increment: The 2 counters in Example 2 failed to support top-2 queries for the data given. This type of problem may occur in every cell if the number of counters is fixed and too small. In the new approach, we increase counters when needed. Thus, the summaries can be initialized with a low number of counters, and more counters can be introduced in cells when needed. Cells with low activity and skewed term occurrence will use few counters, while cells with high activity and temporal variation in the terms will use many counters.

It is important to capture all terms since a miss may lead to an incorrect top-k result. Thus, we perform what we call *Aggressive Increment*. The procedure is covered in Algorithm 4.

Term counts are recorded in a summary until all available counters are utilized, as shown starting in Line 6. The counts are maintained for each processed term, as shown in Line 18. Assume we start with 2 counters, i.e., $limC = 2$. Using the data in Example 2, we are out of counters when we reach the sixth term and need a new counter.

When all counters are used and a new counter is needed for a new term, we find the term with the minimum count as seen in Line 8. If the new term does not influence the targeted top-k result, $q.k \leq targeted\text{-}k$, we proceed as in the algorithm SpaceSaving to take over the counter from the currently least frequent term and record the new term's maximum possible count and maximum possible error (Lines 14–17).

Algorithm 4: bool `insert`(item i)

```
1  T ← currently monitored items ;
2  limC ← maximum #counters ;
3  eFree ← all-counts-error-free flag ;
4  targeted-k ← the targeted k for top-k queries ;
5  if i ∉ T then
6      if |T| < limC then T ← T ∪ {i} ;          // c_i ← 0
7      else
8          j ← min_{j∈T} c_j ;                  // item with min count
9          if c_{targeted-k} < c_j + 1 then
10             if ! eFree then return false ;   // checkpoint
11             limC ← limC × 2 ;                // aggressive increment
12             T ← T ∪ {i} ;                    // c_i ← 0
13         else                                 // else top-k is high enough
14             c_i ← c_j ;                      // previous min count
15             Δ_i ← c_j ;
16             T ← T ∪ {i} \ {j} ;
17             eFree ← false ;                  // error introduced
18 c_i ← c_i + 1 ;                              // increment the counter
19 if c_{targeted-k} < c_i & Δ_i ≠ 0 then
20     // targeted-k items are not error-free anymore, so restore:
21     rollback(i) ;
22     return false ;                           // checkpoint
23 else return true ;
```

However, if the new term influences the top-k result and the summary does not contain errors, the number of counters in the summary is doubled (Line 11). This prevents the term from entering the top-k result with an error. In Example 2, the summary would be doubled from 2 to 4 counters when term *storm* is to be inserted, thus allowing the term to be counted without any errors being introduced into the top-2 result. It is important to maintain error-free top-k results for all cells since we merge cells and summaries when computing query results. With error-free results for the cells, the top-k result of a query is more likely to contain many guaranteed top-k terms: in terms of the query definition in Section II-A, k_g is close to k. We return to this in Section III-C.

When a new term is to be inserted that may influence the top-k, possible errors may have been recorded for counters outside the top-k. As seen from Line 10, we do not double the number of counters when this occurs. Doubling the number of counters when errors exist, the new terms will not maintain any guarantees about previously inserted terms, leaving the summary in an inconsistent state. Instead, we avoid this inconsistent state where new terms have wrong counts by performing a *checkpoint*, to be described in Section III-B3.

When the occurrence of a term that already exists in the summary is to be captured, the associated counter is incremented as shown in Line 18. If the term is outside the top-k counters, the term's counter may record a possible error. When the counter is incremented, it may enter the top-k counters along with its possible error, as seen in Line 19. To prevent this, a *rollback* restores all counters to their state before the term was inserted, and a *checkpoint* is performed. Thereby, the targeted top-k counters remain error-free.

Lemma 1. *The incrementing of counters preserves existing counts and their associated possible errors.*

Proof. Before the number of counters is doubled, no errors exist in the summary according to the algorithm. Thus, no items have been lost and all counters are exact. Therefore, the introduction of new counters has no impact on already inserted items. □

Lemma 2. *For all elements e_i where $i \leq k$, $\Delta_i = 0$.*

Proof. The insertion algorithm allow possible errors to be associated with elements e_i, $i > k$. Before an element with an associated possible error can become an element e_i, $i \leq k$, a *checkpoint* is performed, yielding a new empty summary. Therefore, no element e_i, $i \leq k$ can have an associated possible error. □

The ability of the insertion procedure to increment the number of counters in a summary allows us to start with summaries with few counters. We introduce a parameter *initCounters* to control the initial number of counters. The experimental evaluations study the impact of different settings of this parameter.

3) Checkpointing: To prevent an inconsistent state and a top-k result with a possible error, we introduce a *checkpointing* mechanism. A *checkpoint* is performed before an inconsistent state is reached. The currently active summary is rendered inactive and is archived in its consistent state and is replaced by a new active summary with twice as many counters. Thus, earlier changes to the size of the archived summary due to *Aggressive Increment*, if any, are retained in the new summary, and since that number of counters did not suffice, it is doubled.

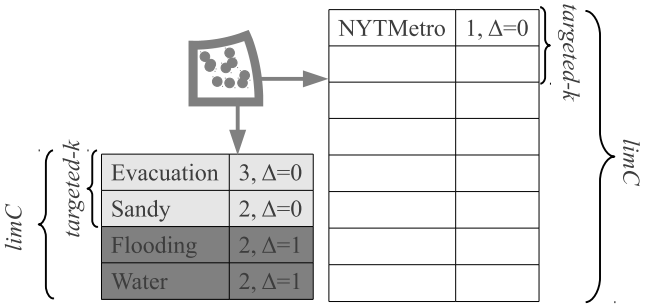


Fig. 7: Cell with One Archived and One Active Summary.

Due to the introduction of *checkpointing*, each cell can contain multiple archived summaries. However, at any point in time, only one summary is active and is maintained as seen in Figure 7, where the left summary is achieved and the right is active.

Example 3. Assume the same setting as in Example 2 and that the terms are processed according to Algorithm 4. When reaching term *storm*, the number of counters is doubled, yielding this summary: $\{evacuation = 3\ (\Delta = 0), sandy = 2\ (\Delta = 0), storm = 1\ (\Delta = 0)\}$. Then, inserting terms *hurricane*, *flooding* and *water* yield: $\{evacuation = 3\ (\Delta = 0), sandy = 2\ (\Delta = 0), flooding = 2\ (\Delta = 1), water = 2\ (\Delta = 1)\}$.

Inserting the term $NYTMetro$ triggers a $checkpoint$ (as seen from Line 10) before errors are introduced in the targeted top-k result. The resulting summaries are shown in Figure 7. □

4) Compaction of Inactive Summaries: Summaries can become inactive for two reasons: (i) due to a $checkpoint$ as just described and (ii) because their time interval ceases to overlap with the current time so that no updates will apply to them. As described in the following section, only the targeted top-k counters are considered during query processing. As part of the archiving of inactive summaries, we thus remove the counters c_j with $j > targeted\text{-}k$. This releases memory that is no longer needed. In the setting of Example 3, the two last grey counters of the left summary in Figure 7 are removed from the summary before archiving.

5) Relaxed Decrement: The number of updates that apply to a cell may vary across time. The *Aggressive Increment* procedure effectively handles the case of increasing activity. However, activity may also decrease, causing already allocated counters become unnecessary and resulting in poor memory utilization. There is thus a need to be able decrement the number of counters in a summary.

Care must be taken when decreasing the number of counters since an overly aggressive decrease may lead to more $checkpoints$. When performing $checkpoint$ too frequently, query performance is reduced because more merging of summaries is needed.

The *Relaxed Decrement* procedure described in Algorithm 5 is designed to reduce the number of counters in a summary without causing excessive $checkpoints$. The procedure is invoked after any given time granule has passed. As shown in Lines 4–5, the number of counters is halved if the summary's number of error-free counters is at least double $targeted\text{-}k$. Then the summary maintains many more error-free counters than required. Note that if any $checkpoint$ was created during the previous time granule, this means that the number of counters has been increasing. Therefore, the activity level may be expected to remain at the same level or continue to increase. Consequently, no reduction is performed.

Algorithm 5: Summary `RelaxedDecrement(int cp, Summary s)`

```
1 cp ← #checkpoints in the previous time granularity ;
2 s.limC ← maximum #counters ;
3 s.errorfree ← #counters without error ;
4 if cp == 0 & s.targeted-k > s.errorfree × 2 then
5     s.limC ← s.limC / 2 ;
6 return s ;
```

C. Query Processing

1) Overview: The spatio-temporal range of a query generally covers a number of cells. We cover a query region and interval with the fewest possible cells from the multi-granularity structure and use the summaries of these cells to produce the query result. Specifically, any query region can be covered by a unique set of most coarse cells. Starting from the coarsest granularity it is checked if each cell is contained in the query region. We refine a cell that merely intersects the query range by the cells it covers at the next, finer granularity until the cells considered are contained in the query region. If a query region partially overlaps other cells, we "snap" the boundary of the query region to the nearest inclusive border. Next, as shown in Figure 6, we proceed to find the summaries that overlap with the query time interval. This results in a number of summaries that are to be "merged" to produce the query result.

The summaries of the cells needed to cover the query range are used to produce the result. This calls for a technique to merge such summaries into a query result (in Line 5 in Algorithm 3). Also, if a query region and interval are identical to those of an existing cell, the same merging procedure applies to possibly several archived summaries associated with that cell.

2) Merging Dynamic Summaries: Recall that each summary maintains counters for only the most frequent terms seen in that summary. Thus, different summaries maintain counters for different sets of terms. Figure 5 illustrates how a wrong result can be produced when merging summaries. The problem occurs when terms that do not occur in all summaries make it into the merged summary.

However, we can still provide guarantees about the merged summaries. Recall that $targeted\text{-}k$ counters of each summary are without any errors. For each summary, we maintain $targeted\text{-}k$+1 counters to provide guarantees about the merged summary. Thus, before running Algorithm 4, the value of $targeted\text{-}k$ is incremented with one more counter. Algorithm 6 details the merging procedure for dynamic summaries.

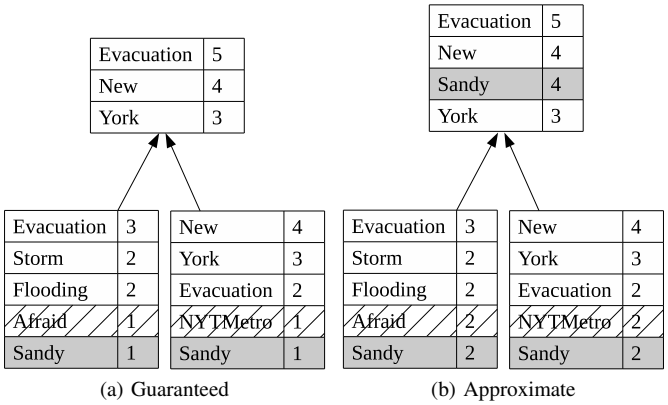
We can guarantee that a merged summary is correct if no other term can possibly make it into the summary. To do this, we utilize the extra counter maintained for each summary—see Line 4. These extra counters provide information about the best possible counts of terms outside the top $targeted\text{-}k$ terms. The best case for the remaining terms is calculated by adding up the values of these counters in each summary. If the resulting value is below the lowest count for a term in the merged summary, the terms are not competitive, and the summary is correct—see Line 21.

In the example in Figure 8a, $targeted\text{-}k$ is 3, the shading identifies the $targeted\text{-}k + 1$st counter maintained by a summary, and the last, grey rows are not stored in the summaries, but are only included for illustration. In this example, the merged summary is correct because no other term can have a count that places it among the top $targeted\text{-}k$ terms. A count that exceeds 3 is needed to enter the merged summary, and the best possible count of a term not in the merged summary is 3.

We cannot always return a correct merged summary. In the example in Figure 8b, the merged summary fails to include the term "Sandy" that has a higher count than "York." The $targeted\text{-}k + 1$st counters maintained by the two cells tell that a term not in the merged summary, e.g., "Sandy," may be able to gather a score of 2 from each cell. The counters guarantee that terms in the merged summary with scores no less than $2 + 2 = 4$ are correct. See Line 27 in the algorithm.

Algorithm 6: $[R, k_g]$ `Merge(int k, parResults P)`

```
1  M ← set used for merging ;
2  foreach p ∈ P do
3      p.T ← terms in the result summary ;
4      p.lc ← the best count outside targeted-k ;
5      foreach t ∈ p.T do
6          t.count ← frequency of the term ;
7          if t ∈ M.terms then
8              M[t].count ← M[t].count + t.count ;
9          else
10             sumlc ← 0 ;
11             foreach p_s ∈ P do
12                 if t ∈ p_s.T then
13                     continue ;
14                 sumlc ← sumlc + p_s.lc ;        // best case
15             t.count ← t.count + sumlc ;
16             t.Δ ← sumlc ;
17             M[t] ← M[t] ∪ {t} ;
18     low ← low + p.lc ;
19 R ← k most frequent terms from M ;
20 if R.minCount ≥ low & ∀t ∈ R (t.Δ = 0) then
21     return [R, k] ;                             // they are guaranteed
22 else
23     G ← set used for guaranteed terms ;
24     A ← set used for approximate terms ;
25     foreach t ∈ R do
26         if t.count ≥ low & t.count - t.Δ ≥
           R.next().count then
27             G ← G ∪ {t} ;
28         else
29             A ← A ∪ {t} ;
30     return [G ∪ A, |G|] ;
```



(a) Guaranteed (b) Approximate

Fig. 8: Guaranteed and Approximate Merging of Summaries

In the example, we return a result with 3 terms along with the information that the 2 first are guaranteed to be correct (while the last may be correct). The correctness of the Algorithm 6 is given in Theorem 1. The experimental evaluation studies the correctness of the merged summaries.

Theorem 1. *Let a query $Q = (R, k, I)$, $k \leq$ targeted-k be given that covers dynamic summaries S_1 and S_2 that describe datasets D_1 and D_2, respectively. Then there does not exist a term t such that $\forall e \in Merge(k, \{S_1, S_2\})_g$ $(score(t, (D_1 \cup D_2) \cap Merge(k, \{S_1, S_2\})_g > e.count)$, where $Merge(k, \{S_1, S_2\})_g$ contains only the guaranteed elements. Further, $Merge(k, \{S_1, S_2\})_g$ is ordered descendingly on the count of the item.*

Proof. It follows from Lemmas 1 and 2 that the counts in S_1 and S_2 are exact and have no errors for the first $targeted\text{-}k$ terms. Merging the summaries produces up to $2targeted\text{-}k$ terms.

There are two possibilities for the count and order guarantees of the merged summary:

First, the counts and order of the merged summary can be fully guaranteed. If the sum of the counts of the $targeted\text{-}k+1$-st terms in S_1 and S_2 is lower than the count of each of the terms in the merged result, the merged summary contains the terms with the highest counts. Also, if the merged summary contains only $targeted\text{-}k$ terms, all the terms are known to be in both S_1 and S_2. Thus, no better combination of terms can exist.

Second, the order of the first k_g, $k_g \leq targeted\text{-}k$, terms can be fully guaranteed. If the k_g-th term has a higher count than the sum of the counts of the $targeted\text{-}k+1$-st terms in S_1 and S_2, the term's count exceeds that of any terms not in the merged summary. Thus, the terms in the merged summary are the ones with highest counts. The count of any term, $t_i, i \leq k_g$ that only occurs in one of the summaries, S_1 or S_2, is increased by the count of the $targeted\text{-}k + 1$-st item of the other summary. If the ordering of the first k_g terms remains after the increments, no terms can exist in D_1 or D_2 that can change the order of the first k_g terms. □

IV. Experimental Evaluation

We proceed to evaluate the AFIA framework and also compare with other available techniques.

A. Experimental Setup

We use two different machines for the experimental study. For small-scale experiments (see Section IV-D), we use a local machine (termed *local*): a quad-core Intel Core i7-3770 with 8 hardware threads and 16 GB of main memory. For large-scale experiments (Sections IV-E–IV-G), we employ an Amazon EC2 High-Memory Cluster (termed HMC) backed by a dual Intel Xeon E5-2670 processor with 32 hardware threads and 244 GB of main-memory. In all experiments, the processing occurs in-memory.

The framework is implemented in Java and we utilize the available parallelism on the machines as follows. One hardware thread is dedicated to a concurrent garbage collector and one to accepting incoming messages. The remaining hardware threads are assigned to equal-sized partitions of the world, as constructed for this purpose.

For the spatial partitioning, we use the Military Grid Reference System (MGRS) [13]. Thus way, we represent any location on the surface of the Earth at multiple granularities, ranging from 1×1 m^2 to 100×100 km^2, by an alphanumeric value. For temporal partitioning, we use the typical temporal hierarchies from data warehousing (hour, day, week, month). We drop cells with very low activity, specifically cells where the most popular term occurs in fewer than 5 messages at the finest granularity (hour). We will shortly describe how we fix the spatio-temporal hierarchy in AFIA based on the considered workload.

We perform three sets of experiments. The first measures the space and accuracy of the proposed algorithms and compares them with two baselines. The second evaluates the throughput, memory consumption, and number of checkpoints when varying the $targeted\text{-}k$ parameter. The third evaluates the query processing performance when varying the size of the spatio-temporal query range. We also show the number of checkpoints created during a time period.

B. Data and Queries

We collected all geo-tagged messages from the public Twitter FireHose during May, 2013. The dataset contains 110,426,053 tweets, yielding an average rate of circa 41 tweet per second. Figure 9 provides insight into the workload and helps choose an appropriate spatio-temporal hierarchy.

Figure 9a shows the total number of cells that receive at least one tweet. Thus, the graph shows the expected number of active summaries at the given granularities. For example, every hour, the number of 0.1×0.1 km^2, 1×1 km^2, 10×10 km^2, and 100×100 km^2 spatial grid cells that receive at least one tweet is circa 0.5M, 200K, 40K, and 10K, respectively (i.e., the "Hour" curve in the figure).

Figure 9b considers the cells at the 0.1 km$\times$0.1 km$\times$1 hour granularity. The y-axis depicts the number of such cells subject to different loads (the number of terms t processed). This shows that the number of "interesting" cells (with at least 100 terms) is quite low, barely reaching 100. Therefore, we fix the finest temporal granularity at an hour.

Figure 9c considers the cells at the 100 km$\times$100 km$\times$1 week granularity. The figure shows that thousands of such cells have summaries that receive more than 1 thousand terms.

We fix k at 25 and generate 2 query sets each consisting of 100 queries. Each query in the first set has a temporal extent with a length chosen at random among 1 hour, 1 day, and 1 week, and it has a randomly selected spatial region. The spatial granularity of the coarsest cell that is contained in the spatial query region is the query's spatial granularity, and the query region is enlarged to cover all the cells at that granularity that it intersects with. In visual terms, this procedure corresponds to "snapping" the spatial query region to the coarser granularity. The underlying rationale for adopting this procedure is that finer-granularity cells do not contribute with term counts at the same magnitude as do cells at the granularity of the coarsest fully covered cell. Since any rectangle can be created from a number of merged squares, we limit the experiments to squares. Each cell contains 10×10 cells at the next, finer granularity. Thus, the largest number of cells any query square can cover is $18 \times 18 = 324$. As a result, we employ queries that may cover up to 1,800 $\times$ 1,800 km^2.

While the first query set focuses on variation in the spatial range, the second focuses on temporal variation. Each of the 100 queries has a spatial region chosen at random from one of the spatial granularities. The query resulting in most temporal summary merges is the query that spans from the first hour after midnight on the first day in May until the hour before midnight on the last day of May. This query covers 2×23 hours, $3+5$ days, and 3 weeks, which amounts to 57 temporal summaries.

To avoid empty results, we do not consider regions that only cover empty cells. Note, that we do not perform compaction on the last (active) temporal granularities, to better reflect the real-life scenario where a stream is constantly keeping these summaries active. To simplify the experiments, the $initCounters$ parameter, which determines the initial number of counters is set to the same value as $targeted\text{-}k$.

C. Baselines

We consider two baselines for comparison and validation of our proposal.

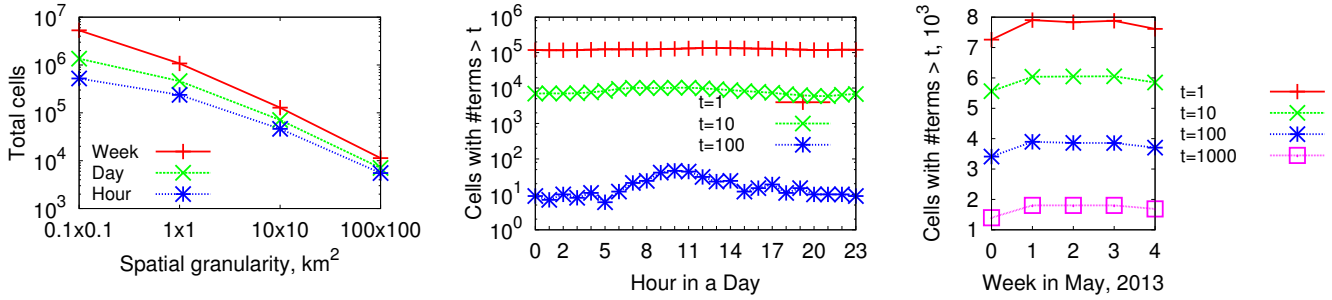
SS: In each active spatio-temporal summary, we aggregate the most frequent items using the SpaceSaving algorithm. Active summaries maintain a fixed number (m) of items and their counts, while archived summaries are compacted by keeping only the counters for the k most frequent terms. Choosing a value for m represents a trade-off between storage size and result accuracy.

HT: In each active spatio-temporal summary, we maintain a full list of terms using a hash table. Once the summary becomes inactive and is archived (its time granularity passes), the list is sorted and cleaned up by discarding all but the counters for the targeted top-k terms. This approach maintains the exact top-k items at all spatio-temporal granularities and serves to measure the accuracy of the other approaches. However, since HT has excessive storage requirements, we do not expect it to scale under the considered workloads.

D. The Space Versus Accuracy Trade-Off

In the first set of experiments, we evaluate the accuracy of our approach by comparing it with the baselines. Since HT maintains the exact counts at all granularities, its storage requirements are too high for large scale experiments. Similarly, SS maintains a fixed number m of counters at all granularities, and m cannot be set too high for large scale experiments. Thus, we consider only tweets occurring in the spatial region with the `11S` grid zone designator (a region in the US of size ca. 900×550 km^2). We use the *local* machine.

We consider variants of SS where m is set to 50, 100, and 200. In AFIA, we set $targeted\text{-}k$ to 50. The first week (four days in May) is used for warm-up, and only the accuracies of the subsequent summaries are evaluated. We perform top-k queries on all summaries at each spatio-temporal granularity. We compare the top-k most frequent terms in each summary produced by AFIA and SS and compare with the corresponding top-k items produced by HT. We set $k = 20$.



(a) Total number of active cells. (b) Terms per hour per 0.1×0.1 km^2 cell. (c) Terms per week per 100×100 km^2 cell.

Fig. 9: Twitter Workload Intensity at Different Spatio-Temporal Granularities

The results are shown in Figure 10. We measure the fraction of correct top-k for varying spatial granularities, and we report separate graph for each temporal granularity. The coarser spatio-temporal summaries are subject to higher loads and more diverse terms. At the finest granularity (0.1×0.1 km^2 in Figure 10a), all approaches produce quite accurate results. However, as the spatial or temporal granularity gets coarser, the accuracy of SS deteriorates. Increasing m does not solve the problem, but just postpones it. On the other hand, AFIA is virtually as accurate as HT at all spatial and temporal granularities. Its *Aggressive Increment* procedure is effective in making it adapt its number of counters as needed to capture the most frequent items very accurately. Due to checkpointing (when several summaries have to be merged), few top-k terms are missed, and we did not observe accuracies below 0.99, 0.98, and 0.97 for hours, days, and weeks, respectively.

Figure 11 reports the average number of counters maintained per summary in the same experiment. Note the log-scaled y-axis. The figure confirms the impracticality of using HT in large scale scenarios. SS maintains a fixed number of counters. The figure shows how AFIA adapts. At hours and days (Figures 11a–11b), AFIA follows SS ($m = 100$), implying that on average 100 counters is enough to achieve the desired accuracy. At weeks (Figures 11c), the adaptivity of AFIA is even clearer: for finer cells, it uses much fewer counters than for coarser cells.

This experiment shows how AFIA achieves the best of two worlds: accuracy is at the level of HT, which is 100% accurate, while the space requirements are at the level of SS with a small number of counters. Having demonstrated the accuracy of our approach, we proceed to large scale performance experiments on the HMC machine.

E. Varying $targeted$-k

In this experiment we vary $targeted$-k and consider two cases where the finest spatial granularity is 0.1×0.1 km^2 and 1×1 km^2. For each case, we also maintain all the coarser granularities. Figure 12 shows the results. AFIA always has a throughput of more than 13,000 items per second (Figure 12a). As expected, the throughput increases as either $targeted$-k decreases or the finest granularity becomes more coarse.

Figure 12b shows how the memory consumption increases with increasing $targeted$-k. This is because increasing numbers of counters have to be maintained to guarantee correctness when there are more terms. The memory consumption increases slowly at first, due to fewer checkpoints occurring (Figure 12c). Thus, for small values of $targeted$-k, much of the space consumption is due to checkpoints. As expected, more memory is required for maintaining finer granularities. The results show that a single HMC machine can maintain between 2 and 3 months of data, meaning that it takes 4 to 6 machines to support a full year. At current costs[1], this is feasible for larger companies. Note that the measured memory consumption represents the entire Java heap memory used by the application. The actual aggregates (not shown) require much less.

When $targeted$-k is set to a low value, the number of required counters will most often also be low. With only a few counters, any small change in activity of a cell may impact the $targeted$-k counters, resulting in a checkpoint. Therefore, we see that the number of checkpoints decreases with $targeted$-k. Having a high number of counters makes a cell more robust to changes in activity, and any small changes can be captured without checkpoints. The number of checkpoints increases as expected when a finer granularity is maintained.

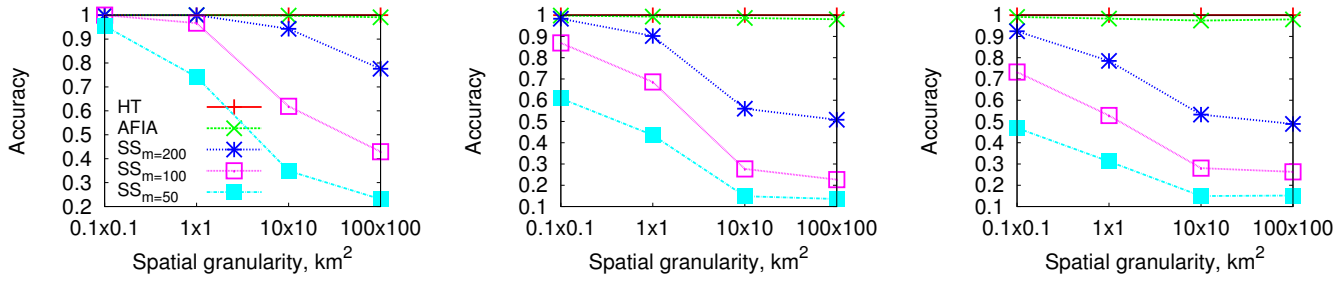
F. Varying $Q.R$ and $Q.T$

We fix $targeted$-k at 100 and vary the spatial and temporal extents of the queries. We employ the two query sets and measure the average runtime. Figures 13a and 13b show the results.

Retrieving a summary requires one or more constant time lookups (in Algorithm 3, Line 3), depending on the spatial or temporal granularity of the query, and it may also require the merging of checkpointed summaries. Because finer granularity cells generally receive fewer items than coarser granularity cells, the runtime is lower at finer granularities. Fewer merges are required because more cells are empty. The results show that this applies to both the spatial and the temporal granularities.

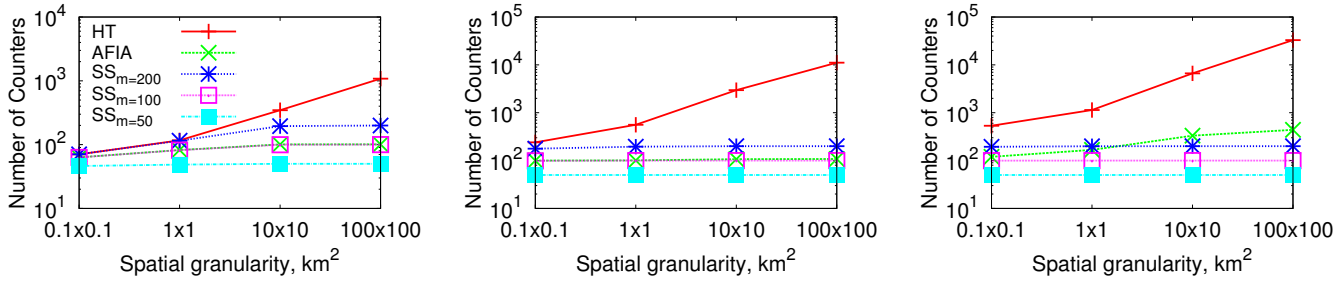
When the spatial extent increases, more lookups are performed, resulting in an expected higher runtime. With more coarse temporal granularities, more merges are performed, increasing the runtime further. The runtime is not made worse

[1]The price for a Spot Instance in June 2013 was USD 0.343 per hour.



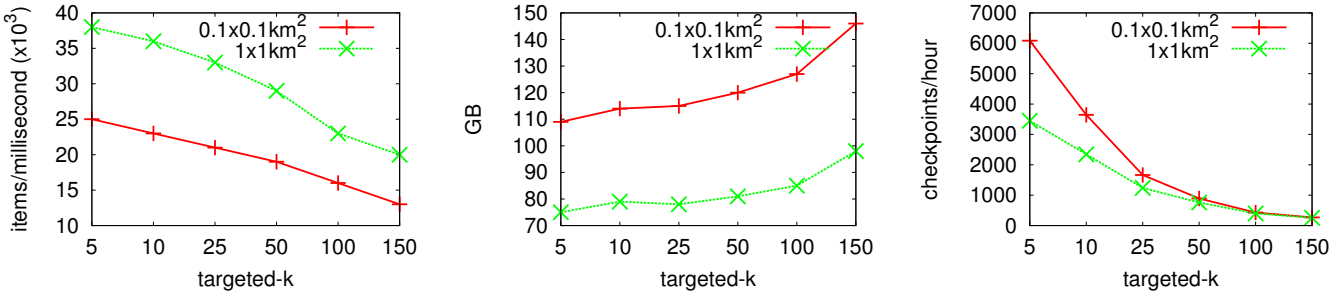
(a) Hours (b) Days (c) Weeks

Fig. 10: Accuracy at Different Spatio-Temporal Granularities



(a) Hours (b) Days (c) Weeks

Fig. 11: Average Number of Counters Maintained at Different Spatio-Temporal Granularities



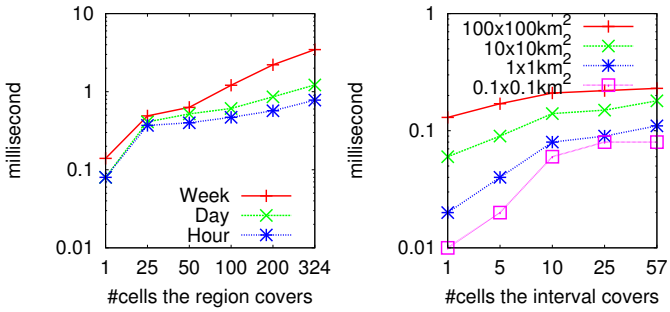
(a) Throughput (b) Memory Consumption (c) Number of Checkpoints

Fig. 12: Varying *targeted-k*

to the same degree when varying the temporal extent since the temporal lookups are performed after the summaries are filtered spatially. Thus, significantly fewer entries exist, resulting in faster lookup times. However, as the spatial extent increases, the runtime also increases since more cells are retrieved and merged.

G. Checkpoints Over Time

In the last experiment, we explore the checkpointing activity over time. We fix *targeted-k* at 100 and count the number of checkpoints per hour for the first 5 days. Figure 14 shows the results. The first day sees more than 18,000 checkpoints, but the number of checkpoints drops quickly. This is because a new day is initialized based on the ending state for the previous day. On the fifth day, only half as many checkpoints are performed. Overall, the number of checkpoints keeps decreasing, but may increase occasionally. This demonstrates the adaptivity of our approach, as the number of counters in the cells adjust to the activity levels of the cells.



(a) Varying Spatial Extent (b) Varying Temporal Extent

Fig. 13: Runtime, Varying Spatial and Temporal Query Extents

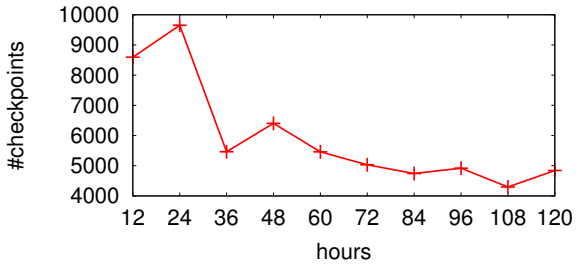


Fig. 14: Checkpoints Over Time

V. Conclusions

We present a new framework for the processing of spatio-temporally constrained top-k most popular terms queries on streaming, spatio-temporally tagged text content, such as microblog posts. The framework's index extends existing techniques for counting frequent items in summaries to allow the summaries to grow and shrink dynamically to adapt to changes in the incoming data. Query processing works by merging relevant summaries. To provide guarantees about the query results, a new merging algorithm is proposed that supports spatio-temporal query regions. Experimental studies with an implementation of the framework offers insight into the accuracy, scalability and performance of the processing of incoming data and queries; and they demonstrate that the framework is capable of offering very accurate query results at high performance at scale.

Acknowledgments

This research was supported in part by the MADALGO research center and in part by the Geocrowd Initial Training Network, funded by the European Commission as an FP7 Peoples Marie Curie Action under grant agreement number 264994.

References

[1] P. K. Agarwal, G. Cormode, Z. Huang, J. Phillips, Z. Wei, and K. Yi. Mergeable summaries. In *PODS*, pages 23–34, 2012.

[2] W. Akman, W. R. Franklin, M. Kankanhalli, and C. Narayanaswami. Geometric computing and uniform grid technique. *Computer-Aided Design*, 21(7):410–420, 1989.

[3] A. Angel, N. Koudas, N. Sarkas, and D. Srivastava. What's on the grapevine? In *SIGMOD*, pages 1047–1050, 2009.

[4] N. Bansal and N. Koudas. Blogscope: a system for online analysis of high volume text streams. In *VLDB*, pages 1410–1413, 2007.

[5] E. Baralis, S. Paraboschi, and E. Teniente. Materialized view selection in a multidimensional database. In *VLDB*, pages 156–165, 1997.

[6] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *ICALP*, pages 693–703, 2002.

[7] Z. Cheng, J. Caverlee, and K. Lee. You are where you tweet: a content-based approach to geo-locating twitter users. In *CIKM*, pages 759–768, 2010.

[8] G. Cormode and M. Hadjieleftheriou. Finding frequent items in data streams. *PVLDB*, 1(2):1530–1541, 2008.

[9] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[10] G. Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. *TODS*, 30(1):249–278, 2005.

[11] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *AlgorithmsESA 2002*, pages 348–360, 2002.

[12] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[13] J. W. Hager, J. F. Behensky, and B. W. Drew. The universal grids: Universal transverse mercator (utm) and universal polar stereographic (ups). Technical report, 1989.

[14] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD Record*, volume 25, pages 205–216, 1996.

[15] P. Kapanipathi, F. Orlandi, A. Sheth, and A. Passant. Personalized filtering of the twitter stream. In *SPIM Workshop at ISWC*, pages 6–13, 2011.

[16] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *TODS*, 28(1):51–55, 2003.

[17] R. Lan, M. D. Lieberman, and H. Samet. The picture of health: map-based, collaborative spatio-temporal disease tracking. In *Workshop on Use of GIS in Public Health*, pages 27–35, 2012.

[18] I. Lazaridis and S. Mehrotra. Progressive approximate aggregate queries with a multi-resolution tree structure. In *SIGMOD*, pages 401–412, 2001.

[19] M. D. Lieberman and H. Samet. Supporting rapid processing and interactive map-based exploration of streaming news. In *GIS*, pages 179–188, 2012.

[20] M. D. Lieberman, H. Samet, J. Sankaranarayanan, and J. Sperling. Steward: architecture of a spatio-textual search engine. In *PODS*, page 25, 2007.

[21] N. Mamoulis, S. Bakiras, and P. Kalnis. Evaluation of top-k OLAP queries using aggregate R-trees. In *SSTD*, pages 236–253, 2005.

[22] N. Manerikar and T. Palpanas. Frequent items in streaming data: An experimental evaluation of the state-of-the-art. *DKE*, 68(4):415–430, 2009.

[23] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, pages 346–357, 2002.

[24] M. Mathioudakis, N. Bansal, and N. Koudas. Identifying, attributing and describing spatial bursts. *PVLDB*, 3(1):1091–1102, 2010.

[25] M. Mathioudakis, N. Bansal, and N. Koudas. Identifying, attributing and describing spatial bursts. *PVLDB*, 3(1-2):1091–1102, 2010.

[26] M. Mathioudakis and N. Koudas. Twittermonitor: trend detection over the twitter stream. In *SIGMOD*, pages 1155–1158, 2010.

[27] M. Mathioudakis and N. Koudas. Twittermonitor: trend detection over the twitter stream. In *SIGMOD*, pages 1155–1158, 2010.

[28] A. Metwally, D. Agrawal, and A. El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *ICDT*, pages 398–412, 2005.

[29] J. Misra and D. Gries. Finding repeated elements. *Science of computer programming*, 2(2):143–152, 1982.

[30] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP operations in spatial data warehouses. In *SSTD*, pages 443–459, 2001.

[31] J. Sankaranarayanan, H. Samet, B. E. Teitler, M. D. Lieberman, and J. Sperling. Twitterstand: news in tweets. In *GIS*, pages 42–51, 2009.

[32] D. Sidlauskas, S. Saltenis, C. W. Christiansen, J. M. Johansen, and D. Saulys. Trees or grids?: indexing moving objects in main memory. In *GIS*, pages 236–245, 2009.

[33] K. Tao, F. Abel, Q. Gao, and G.-J. Houben. Tums: Twitter-based user modeling service. In *The Semantic Web: ESWC 2011 Workshops*, pages 269–283, 2012.

[34] B. E. Teitler, M. D. Lieberman, D. Panozzo, J. Sankaranarayanan, H. Samet, and J. Sperling. Newsstand: a new view on news. In *GIS*, page 18, 2008.

[35] J. Vosecky, D. Jiang, and W. Ng. Limosa: a system for geographic user interest analysis in twitter. In *EDBT*, pages 709–712, 2013.

[36] J. Yang and J. Widom. Incremental computation and maintenance of temporal aggregates. In *ICDE*, pages 51–60, 2001.