Efficient Geospatial Analytics on Time Series Big Data

Isam Mashhour Al Jawarneh, Paolo Bellavista , Antonio Corradi, Luca Foschini , Rebecca Montanari

Dipartimento di Informatica – Scienza e Ingegneria, University of Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
{isam.aljawarneh3, paolo.bellavista, antonio.corradi, luca.foschini, rebecca.montanari}@unibo.it

***Abstract*— In smart city advanced analytical scenarios, tremendous amounts of georeferenced big time series data arrive continuously to time series databases, requiring the shared analytics on both geospatial and time dimensions. Mostly, the focus has been given to optimizing the storage and processing of each workload alone, either geospatial or time dimensions. To close this gap, in this paper, we have designed a pyramid-like indexing scheme that we term as geoTSI (short for geo time series index) which twists two dimensionality reduction geospatial encoding methods (geohash and S2) sequentially with a time series index to efficiently enable such mixed workload scenarios. This method enables geospatial and time indexes to collaborate synergistically in an aim to reduce the time required for accessing the disk and retrieving the time series data that comprises the answer for the mixed workload query. We show how our indexing scheme can be efficiently exploited to run a hybrid geospatial proximity query on time series data. Also, we evaluate our index on real-world georeferenced time series data, where we obtain, on average, a significant 34 % reduction in the query running time by applying our method against the baseline.**

Keywords*— *time series, dimension reduction, geospatial indexing

I. INTRODUCTION

Huge amounts of time-series data are collected daily from various smart city application scenarios. For example, IoT sensors attached to weather stations for collecting weather and environmental data. Several research has been done to index time series data to improve the query access capabilities [1, 2]. However, the focus of those studies has been geared toward indexing the time dimension. Nevertheless, time series data is typically associated with locational data indicating the geographies where they have been collected. Numerous scenarios exist that ask specifically to filter such geo-referenced time series data on both dimensions, the time and location. However, works on indexing both dimensions are still at its infancy. Georeferenced time series data are those records that are associated with locational tags.

Innumerable scenarios in smart cities require the awareness of location dimension while analyzing time series data. For example, to understand the trends of vehicle mobility in different zones of a metropolitan city, a request may require the moving average of the speeds of vehicles, that is the average of the speed in each region across time. In environmental studies, georeferenced times series data can be utilized to capture close-by zones with high similarity in air pollution patterns. That is to say, how air pollutants concentrations develop across time in each zone and weather an autocorrelation exists between nearness and pollution similarity.

Highly performing queries on georeferenced time series data require indexing both dimensions, time, and location. To the best of our knowledge, current works do not focus on scenarios that require a joint analysis on location and time dimensions in series data. Most works focus on indexing the time dimension aiming at improving similarity search queries on time series data. However, in case that a query contains a filter that focuses on the geospatial dimension, the query performance degrades as the location dimension is not appropriately indexed.

In this paper, we show the design and prototyping of a novel efficient geospatial multilayer pyramid-like indexing scheme for time series data. Our novel index consists of two levels. At the first level, we employ a geospatial dimensionality reduction approach known as geohash as a quick filter that retrieves a set of series that fall within a given geospatial proximity to a given time series (we refer to the latter as the 'center time series' hereafter, analogous to a Point-of-Interest, POI in geospatial proximity analytics). We then pass this partial set, which is not yet optimized, to another filter that is more accurate (known as Google's S2) to further prune the first set. Afterwards, we pass the optimized set to the query processor that applies the expensive geometrical operation to exclude a partial set of series (or part of those series) that do not contribute to the query result. In this paper, we focus on utilizing our novel indexing scheme to solve a novel type of mixed workloads geospatial on time series queries that we refer to as 'geospatial proximity on time series'.

The remainder of the paper is organized as follows. We first briefly review the related literature in section II. We then discuss the foundational background relevant for the discussion. Thereafter, we discuss the design of our pyramid-like indexing structure and devise an algorithm for utilizing it in geospatial proximity on time series queries. In what follows, we showcase and discuss our results. We close the paper by conclusions and future research directions.

II. RELATED LITERATURE

Several works in the recent related literature have focused on developing optimized methods for the storage and processing of geospatial big data. For example, the works by [3, 4] have

designed a hybrid indexing scheme for the storage and processing of big geospatial data over a document-oriented de facto database (i.e., MongoDB), but, however, did not consider the time dimension. In a similar vein, works by [5, 6] have focused on building optimizers for geospatial proximity-like queries in main-memory big data processing frameworks such as Apache Spark [7]. However, those systems are only appropriate for the geospatial dimension without an attention to the time dimension.

On the contrary, numerous are the works in the relevant literature that focus solely on the time dimension of the time series. For example, indexing strategies such as ADS+ [1] and iSAX2+ [2] which both attempt to minimize the time required for building and adaptively maintaining the time series indexes aiming at speeding up queries on time series big data. However, those are irrelevant for the shared processing of the geospatial dimension over time series data.

Our methods presented in this paper hybridize two efficient dimensionality-reduction geospatial encoding approaches (i.e., geohash an S2) with an efficient time-series index (i.e., Time Series Index TSI of the InfluxDB) so that they collaborate synergistically in efficiently answering geospatial queries on time series big data.

III. Preliminaries and Theoritical Foundations

In this section, we define a reference data structure for time series data. In addition, we formulate the problem of proximity-like queries over time series data. We also recapitulate an efficient geospatial processing approach known as filter-and-refine, which is relevant for the discussion.

A. Baseline data structure for time series data

To facilitate the discussion that follows, we hereby anchor a baseline data structure that will be used to define the theoretical foundations that follow in the next subsections. A time series is a sequential timely-ordered group of points where each point has a specific value on some attribute. It is typically defined as $S = [V_1, \ldots\ldots, V_k, \ldots, V_n]$ where V_k is the value of measurement at the k^{th} point in time, whereas n is the total number of points in the series.

To be able to store and query huge amounts of time series data efficiently, we presume a storage engine that divides time series data into chunks, where each chunk is a time-bounded subset of S.

For example, $S_1 = [V_1, \ldots\ldots, V_{n_1}]$, $S_k = [V_{n_1+1}, \ldots\ldots, V_k]$, $S_m = [V_{k+1}, \ldots\ldots, V_n]$ is a list of subset series forming altogether S. Formally speaking, $S = \bigcup_{i=1}^{m} S_i$. Each series that belongs to S covers a specific and predefined block of time (e.g., two hours).

To be able to query time series data efficiently, series (subsets of S above) are indexed. Each series S_k is indexed by an additional key/value pair that facilitates its access at query time, which then can be mapped to a series ID, where the latter can be used to access the specific partitions in memory where the series with that ID physically resides. We have selected this structure as it is widely accepted by well-performing de facto time series database systems (TSDB) such as InfluxDB.

B. Geospatial analytics for time-series data

Geospatial analytics involve several kinds of geometrical queries such as proximity searches, *k*NN and geospatial join. In this paper, we focus on the geospatial proximity search queries applied to time series data.

A geospatial proximity search query over time series data can be defined as it follows. Given a georeferenced time series S_{ref} that is geolocated in the geometrical space on Earth, find all series which have a geographical distance that is at most α_{geo} from the location of the S_{ref} reference time series (we refer to the latter as the 'center time-series' hereafter) and are at most α_{tim} time-range distant from the center time series S_{ref}. An example query of such kind can be expressed in an SQL-like prose such as the following: "select avg(speed) from trips where time > now () - 1h and ST_Distance (POINT (latitude, longitude), S_{ref} *.loc*) < radius".

The S_{ref} contains all points in the time series that are located at the reference geolocation S_{ref} *.loc* in the last one hour. Say, $S_{ref} = [V_1, \ldots\ldots, V_k, \ldots, V_n]$, where V_k is the value of measurement at the k^{th} point in time (e.g., 'speed' in the example query), whereas n is the total number of points in the series. The problem then simplifies to finding series (or subset of those) that have a time range which falls within the same time range as S_{ref} (i.e., are $< \alpha_{tim}$) and are at most α_{geo} (radius in the SQL-like query above) geometrically far from the geolocation of S_{ref}.

For a plain version of the query engine that operates without a proper indexing on the geolocation dimension, for each series in the database defined as $S' = [V_1, \ldots\ldots, V_k, \ldots, V_n]$, a time predicate first checks whether this series time range falls within the same range as S_{ref}. If so, the whole series will be returned. All in all, if we refer to the series that satisfies the time predicate as S_i^{valid} thereafter, for this series, the geospatial predicate searches for the values geolocated with the S_{ref} location (i.e., at most α_{geo} far from the center time series). The returned set of series is represented as follows: $S_{tim} = \bigcup_{i=1}^{n} S_i^{valid}$ where n is the number of series in the database that satisfy the time predicate. Then for each S_i^{valid} in S_{tim} the geospatial predicate checks whether the series (or a subset of it) falls within the range specified by the α_{geo} threshold. That is to say,

$S_{geo} = [\bigcup_{i=1}^{n} S_i^{valid\_sub} \mid \forall\ obj_k \text{ in } S_i^{valid\_sub}, \text{dist}(obj_k, S_{ref}.Loc) <= \alpha_{geo}]$.

where S_{geo} is the result set that satisfies the time and geospatial predicates, $S_i^{valid\_sub}$ is a subset of the S_i^{valid} such that each object in that subset (obj_k) satisfies the geospatial predicate (distance between obj_k and the location of the 'center time series' ,defined in the equation as $S_{ref}.Loc$, is less than or equal the geospatial threshold α_{geo}). It is obvious that without a relevant indexing on the geospatial dimension, all points (objects) in each series that satisfies the time predicate will be checked against the geospatial predicate, which easily becomes cumbersome for big time series data. Figure 1 shows a toy example of the geospatial proximity search on time series data.

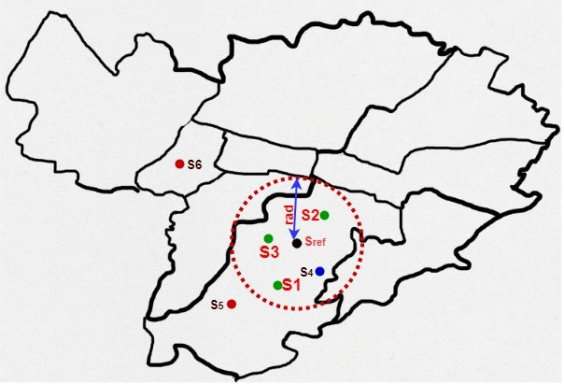

Fig. 1. A heuristic overview of geospatial queries on time series data

As shown in Figure 1, S1, S2, S3 fall within the range of center time series and within the specified geolocation distance (α_{geo}). Despite that S4 falls within the circle, its time series falls outside the time-series range of the 'center time series' S_{ref}, hence outside the result set. Also, despite that S5 has the same time range as S_{ref}, it is located outside the circle, so it does not belong to the result set. S6 falls geometrically outside the circle and its time range is outside that of S_{ref}. so, the result set $S_{geo} = [S1, S2, S3]$.

Calculating the geometrical distance between every object in each series and the location of S_{ref} is computationally expensive for very large datasets. So, in this paper we adapt the filter-and-refinement approach as it will be explained in section IV.C. The total cost of this plain version of geospatial proximity over time series is calculated as it follows. Suppose that the cost of scanning the set of series for selecting those that satisfy the time range predicate (the time range of the S_{ref} series) is C_{tim}. The cost of calculating the geographical distance for all objects in each series that resulted from the first step is $C_{geo} = \sum_{i=1}^{n}\sum_{k=1}^{m} C(\text{dist}(obj_{i,k}, S_{ref}.Loc))$. The total cost is then $C_{total} = C_{geo} + C_{tim}$. Where $\text{dist}(obj_{i,k}, S_{ref}.Loc)$ is the distance between object obj_k in series i and the location of the 'center time series', which is $S_{ref}.Loc$. m is the total number of objects in series i and n is the total number of series that satisfy the time predicate α_{tim}.

It is obvious that the total cost increases significantly as the number of times the method performs a distance calculation increases. Having said that, the goal is to reduce the number of times we perform distance calculations. We will show how we could achieve this in section V by applying our novel methods that are discussed in sections IV.A and IV.B.

C. Filter-and-refine geospatial processing approach

Because performing geometrical calculations on big geospatial data is resource and computation-intensive, well-performing methods adopts the so-called filter-and-refine approach for performing geospatial analytics such as proximity search. As its name implies, it simply operates in two-stages, 'filter' and 'refine'. In the filter stage, a geocode-based covering is imposed on the query geographical area so that it completely covers it. For example, if the geocoding system that is adopted is geohash and the geospatial query is a proximity query that search for points which fall at a specific distance from a point-of-interest (POI), then the query area is a circle, and the covering is termed as the geohash covering. Each geohash can be thought of as a small rectangle that has a string geohash coding where each point in the rectangle will have the same geohash value. The covering is simply a set of geohash values. The 'filter' stage then proceeds by geohash-coding each point in the database and matches those with the set of geohash values in the covering. It then returns all points that interact with the covering (i.e., have a corresponding geohash value in the covering set). But this results in the so-called 'false positives', those are the points that have their geohash values equivalent to one of those in the covering set, but otherwise fall geographically outside the query area (the circle in the proximity search example). The 'refine' stage is then necessary to discard the 'false positives' by applying the exact geometrical operation (e.g., Haversine distance calculation formula) on the partial set that resulted from the 'filter' stage [8]. This way we guarantee that we reduce the number of times we apply the costly geometrical operation. As this is the state-of-art in geospatial analytics optimization, we have adapted it with a simple tweak so that it operates on georeferenced time series big data as will be explained in section IV.

IV. Efficient Indexing for Geospatial Analytics on Time Series Big Data

In this section, we explain our novel efficient algorithms that we have designed for improving the geospatial analytics for time series data. We focus in this paper on geospatial proximity for time series datasets.

A. Geospatial Hybrid Indexing Structure

As we have stipulated in section III.A, geometrical computations that are required for most common geospatial analytics are typically expensive. The same applies for proximity search that requires applying a geometrical distance calculation formula for each object in the database in its plain version. So, in geospatial processing, dimensionality reduction geocoding systems are used such as geohash (a variant of Z-order curves) and Google's S2 (a variant of Hilbert curves).

Each one of those geocoding systems is associated with some advantages and disadvantages. Geohash encoding is computationally cheaper than S2 as it has a linear embedding which acts by dividing a planar version of the earth recursively until a specific precision of the geohash code is reached. The quadratic embedding of the S2 (using the quad-tree data structure) renders it slower than the geohash encoding despite being more accurate.

To take advantage of both worlds without their limitations, we have decided to design a hybrid pyramid-like indexing scheme that we term as geoTSI (short for geo time series index) that utilizes both geocoding systems so that they operate synergistically without their limitations. As indexing is performed on-the-fly in our case (upon receiving the 'geospatial on time-series' query), we first index the dataset using the geohash encoding (the faster portion, level 1 in Figure 2), which results in all the data points geohash-encoded. Thereafter, we add the geohash to the key group, which is the group of keys that will

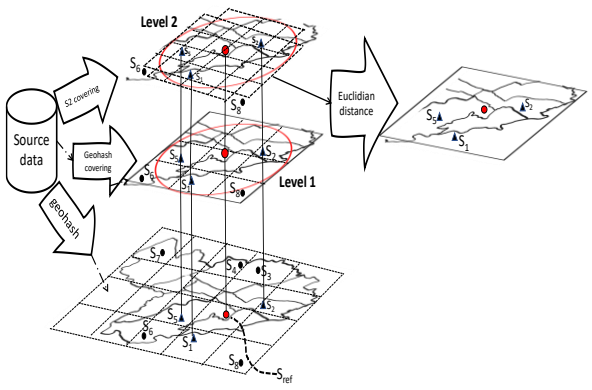

Fig. 2. High level overview of our method for performing geospatial proximity on time series data

be used to divide the data into chunks (i.e., partitions). This way, we guarantee that every set of points which have the same geohash value end up in the same partition (a.k.a. geo co-locality). Also considering that the data is indexed on time dimension so that each chunk covers a predefined time range. Upon receiving a 'geospatial proximity on time series' query, we utilize this first portion of our pyramid-like index to search for a set of series that satisfy a geohash predicate (as will be defined in section IV.B). Geohash in this case acts as a quick-and-dirt sieve that returns true-hit objects (objects that satisfy the geohash predicate and which are located in the locations that they are supposed to represent in real geometries) in addition to few objects referred to as 'false positives' (those are the objects that have their geohashes fall within the list of geohashes in the predicate, but otherwise fall outside in real geometries). The second part of our hybrid pyramid-like indexing strategy depends on the fact that we, at this point, utilize the more accurate (but slower) S2 geocoding. We specifically pass the partial set of series that we obtain from the first stage to an S2 geocoder that generates an S2 cell coding (term it as S2_cell_code) for each object in the partial set (level 2 in Figure 2). We then replace the 'geohash' in the key group with the S2_cell_code and repartition the data so that chunks contain points that belong to the same S2_cell_code bucket and fall within the time range of the chunk. The data in this shape (which is a partial set of the database) is ready to be served to a refinement stage that applies the exact geometrical predicate (e.g., Haversine distance formula in case of proximity queries) on the candidates so that to specify which of them falls within the query area in real geometries as will be explained in section IV.B. In the following subsection, we explain the details of a general algorithm that we have designed for utilizing this novel indexing scheme in answering 'geospatial proximity on time series' queries. The lower two levels of our indexing scheme are shown in Levels 1 and 2 of Figure 2.

B. Effecient Algorithm for Geospatial Proximity Queries on Time Series Data

The proximity geospatial query on time series data that we have described in section III.B can be efficiently solved by our novel algorithm that operates as it follows. We assume the baseline data structure that we have described in section III.A for efficiently storing georeferenced big time-series data. As a short recap, time-series data is stored within partitions in disk, where each partition with a predetermined ID contains one or more series with specific IDs. Each series contains only values of a measurement (e.g., speed) for points that fall within a specific key/value pair bucket (e.g., street segment of a city, where the key is 'street segment ID' and the value is '1'). This means that speed values that are stored in that series all belong to street segment that has the ID which is equal to '1'. Also, each partition covers a predefined time range.

After receiving a 'geospatial proximity on time series' query such as the following one: "find all series that are at most σ geographically far from the center time series", our geospatial 'proximity query on time series' optimizer then proceeds as it follows. Given the maximum distance (i.e., radius), the points we are searching for fall within a circle that has a radius ($say\ r$) which is equal to the maximum distance from the 'center time series'. Our method first encloses the circle within a rectangle, where the width and length of the rectangle each equals to $(2.r)$. Our method then generates a geohash covering that completely covers the rectangle (as shown in Level 1 of Figure 2). Thereafter, we apply a tweaked version of the filter-and-refine method that is described in section III.C as it follows. The 'filter' stage of the tweaked version first selects all the series in all partitions that have a key/value geohash pair that is equivalent to one of those available in the geohash covering of the 'enclosing rectangle' (i.e., interacts with the geohash covering). Lines numbered 1 through 5 of Algorithm 1 covers this procedure. Remind that at this point, data is partitioned to chunks where each chunk contains data that has the same geohash value. Since geohash is an approximation, this step retrieves a set of series that contains 'false positives'. In other terms, those are the series that have their key/value geohash pairs equivalent to some of those in the geohash covering but fall outside the covering in real geometries as we have described in section IV.A. Also remind that since data in each chunk covers a predefined time range, only chunks that satisfy the time predicate will be returned at this stage. This first filtering stage forms the 'speeding' part of our strategy, where we aim to retrieve a quick-and-dirty set of series for further processing. Afterwards, our method imposes an S2 covering for the circle that is enclosed by the 'enclosing rectangle', as shown in level 2 of Figure 2. Our method then shapes the data from the first stage so that it conforms with the newly imposed S2 coverer. That said, it first geocodes each point in each series from the first stage with an S2 geocode, it then repartitions the series so that the partitioning key is the S2 instead of the geohash. Stated another way, all values in each series belong to the S2 geocode that represents the value for the S2 key of the series. It then retrieves the set of series that have their key/value pairs for the S2 which equal to one of those in the S2 coverer. This second filtering procedure is covered in lines numbered 6 through 14 of Algorithm 1. Having completed this second filtering stage, we will be having a set of series with fewer data that discards some of the false positives which resulted from the first filtering stage. After that, our method proceeds by applying the computationally expensive distance calculation on this

partial set to discard all the false positives. This refinement stage is coded in lines numbered 15 through 21 of Algorithm 1. What makes this method unique as compared to the plain version of filter-refine is that it applies two consequent filtering stages, one that is quick-and-dirty (i.e., the geohash) and the subsequent which is slower and more accurate (i.e., the S2), thus benefiting from both worlds without their limitations. Also, the geospatial filtering is performed in consistence with the time predicate as opposed to the plain geospatial version that is discussed in section III.C, which otherwise works for the geospatial dimension only. Algorithm 1 lists the steps by which our optimizer for 'geospatial proximity on time series' search operates.

Algorithm 1. Geospatial proximity on time series

Input: *center time series (CTS), radius (r)*

Output*:* $\bigcup_{i=1}^{n} L'_i$ *where* L'_i *is a series that is at r maximum distance from the location of CTS*

```
1:  coveringGeo <set> = genGeoCoverer (CTS, r)
2:  S' <set> = [], L <set> = [], L'' <set> = []
3:  Foreach series S in database
4:  |   If (contains (coveringGeo, S.geohash)
5:  |   |   S' = S' ⋃ S
6:  Foreach S'' in S'
7:  |   Foreach point in S''
8:  |   |   S2_cell_id = S2geocode(point)
9:  groupKey.add (S2_cell_id)
10 S2Coverer <set> = genS2Coverer (CTS, r)
11: T <set> = Repartition (S', groupKey)
12: Foreach T' in T
13: |   If (contains (S2Coverer, T'. S2_cell_id)
14: |   |   L = L ⋃ T'
15: Foreach L' in L
16: |   N = [] //empty series
17: |   Foreach point in L'
18: |   |   If (distance (CTS, point.loc) < r)
19: |   |   |   N.insert (point)
20: |   L'' = L'' ⋃ N
21: return L'' //the list of series satisfying geospatial and time range filters
```

V. Experimental Evaluation

In this section, we summarize test settings, the datasets, in addition to the baseline method and evaluation metrics.

A. Experimental setup

Datasets. We use a vehicle mobility dataset that consists of around 1155K tuples, representing Electric Taxi GPS mobility trips for one day in the Chinese city of Shenzhen [9] . The target measurement in this data is the 'speed'. Data is transferred to the database using the 'line protocol' of InfluxDB. Within the premises of InfluxDB, data is stored as points. Each point has four parts: timestamp, measurement, 'tagset' and 'tagfield'. Those components have the following analogy when compared with relational database systems: a measurement is analogous to tables, tagset is a key-value pair dictionary that is similar to an indexed key column in relational DB systems, fieldsets are typed scalar values that are analogous to unindexed columns in relational databases. To take a more utilitarian perspective, a data point in our dataset is represented using the line protocol as the following example: "trips, geohash=ws105j, s2_cell_id=3403f154, speed=29, timestamp". In this example: "trips" is the measurement, which is analogous to tables in relational database systems. Also, 'geohash' and its value are 'tag key' and 'tag value', respectively. Also, 's2_cell_id' and its value are 'tag key' and 'tag value', respectively, whereas 'speed' and its value are 'field key' and 'field value', respectively. As for the data structures, data is distributed into shards (a.k.a. partitions), where each shard contains a set of series. Each series has an ID and stores columnar-organized points. At the lowest level (first filtering stage), data is organized and indexed on the geohash value, whereas it is reorganized and reindexed on the S2 value at the second level (second filtering stage). This organization has a plausible impact on the performance of the geospatial on time series queries as mostly those queries are aggregations over time. In other terms, searches for points bounded by time and location predicates, followed by summary functions such as 'average'. It is well-established that columnar databases match favorably those query patterns as opposed to row-based counterparts.

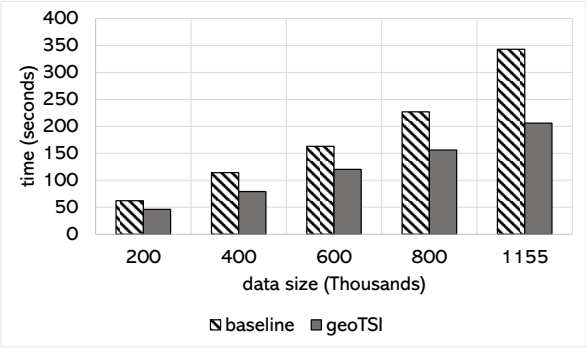

Fig. 3 total time baseline Vs. geoTSI

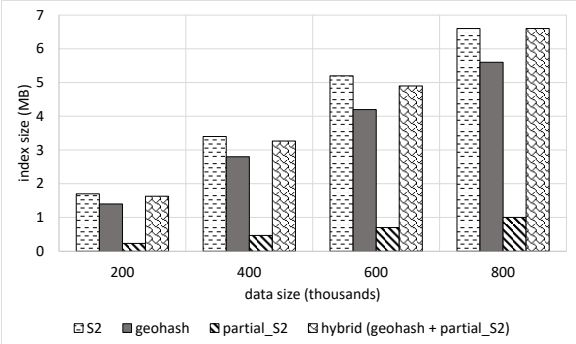

Fig. 4 Index size, solo S2 against hybrid (partial S2 & geohash)

Baselines. We have compared our method with a plain single-index ill-documented experimental baseline that is of InfluxDB which indexes time series data using S2 only.

Parameter setup. To measure the efficiency of utilizing our indexing scheme within our optimizer algorithm for 'geospatial proximity on time series data' queries. We depend on varying the size of data from 200k points to around 1155K points with each step increasing the size by 200k points. To make a fair comparison, we chose a geohash precision that equals to 6 and an S2 precision that equals to 13. Our intention for doing so is the following. At geohash precision that equals to 6, the average area of each rectangle representing a geohash value in the coverer is almost 0.7442 km^2, while the minimum range of the area of the S2 cell at level 13 is 0.76 km^2. We find those to be the nearest possible values when comparing geohash coverers with S2 coverers. Also, we have fixed the maximum distance to 2 kilometers and selected the geolocation of the 'center time series' to be the center point of the city. We choose time predicate that covers 'few hours' time range that is a subset of the total time period covered by the data.

B. Experimental Results

First, varying the data size and computing the total time. In the case of the baseline, total time equals to the time required for applying the time range predicate plus the time required for the index creation (solo S2 in the baseline case) plus the time required to apply the 'filter' stage of the filter-and-refine approach plus the time required to apply the exact geometrical computation (i.e., distance calculation) to discard 'false positives'. In the case of our system (geoTSI), it is equal to the time of indexing using the geohash plus the time required for applying the geohash 'filter' plus the time required for applying the S2 indexing and predicate on the partial subset (which is obtained after applying the geohash) plus the time of applying the distance calculation to discard the 'false positives'. For both cases, the end-to-end time contains the index creation and the query execution times.

Figure 3 shows the results of the total time described above for the baseline against our novel system geoTSI. On average, we obtain a significant 34 % reduction in the total time. Part of this reduction in total time is obtained because the index creation time in the case of the baseline (indexing the whole dataset using the S2 only) is pretty much higher than that of the pyramid indexing (the geohash first then the S2 on a partial set). This is because geocoding using the geohash is cheaper than if S2 is to be used alone as we have explained in section IV.A.

We have also measured the size of each index, geohash alone, solo S2, and the hybrid pyramid-like index. As shown in Figure 4, the size (in megabytes) of our hybrid index (geohash on all data points and S2 on a partial set after applying the geohash filter) is almost on par with the case as if solo S2 was applied.

Counterintuitively, a tiny contributing factor in this reduction in total time is that the total points retrieved by applying the geohash filter alone is slightly less than that of S2, while it is the least when applying the hybrid index (geohash on all the data followed by the S2 on partial list) as shown in Figure 5. This is self explanatorily as depicted by the fact that geospatial data is highly skewed [10] (right-skewed in this case), meaning that more points in the data end up falling within the premises of S2 buckets of the S2 coverer as compared to the geohash coverer or our pyramid stepwise coverer (the geohash coverer followed by the S2 coverer). This is so because a geohash with the selected precision (which equals to 6) has a higher index selectivity as compared to a relevant equivalent in the S2 (equals to 13 in this case).

VI. Conclusions and Future Works

In this paper, we have designed a novel pyramid-like indexing scheme that enables efficient geospatial analytics on georeferenced time series big data. Our indexing method build atop a time series index (TSI) by layering a pyramid-like structure that twists a geospatial two-layer index. Our experiment on a real-world dataset shows the efficiency of our indexing structure when utilizing it to perform 'geospatial proximity on time series' searches that hybridize geospatial proximity searches with time series 'time range' filters. As a future research perspective, we intend to extend our optimizers for other kinds of geospatial on time series queries such as Top-N and geo-spatial join. It seems that our optimizer is easily extensible to the case of Top-N as the way our method works in this paper is part of the Top-N with the base geometrical operation being the Point-in-Polygon (PIP) (in this simple case is a regularly-shaped polygon – a.k.a. circle), thus extending to a more general arbitrarily-shaped polygon should easily follow by applying a simple tweak. As an anticipation, imposing a geohash covering each polygon, then matching all series that interact with the cover for each polygon should do the trick.

In addition, in this paper we focus on comparing a geohash precision that equals 6 to S2 precision that is equal to 13. We intend in the future to expand the experimentation by obtaining more results for other comparable values of geohash and S2 precisions. Also, the solution that we proposed in this paper is experimented with one type of datasets (a.k.a. mobility geospatially-tagged time series data). In a future work, we plan to expand the experiments using other locational time series datasets, such as climate change data, probably fusing them with mobility data to investigate the effect of the fusion of heterogeneous data on the performance of the proposed solution. Such integration would allow for more advanced types of geospatial over time series queries that would be otherwise intractable. A convenient work in this direction from the recent state-of-art appears in [11], where we have designed a novel method for integrating mobility and meteorological data at scale with quality of service guarantees. In that work, we depend on the locational fields for efficiently integrating the heterogeneous mobility and meteorological data. In a future work, we intend to extend that work by integrating the data based on the two dimensions: temporal and locational. That way, we will have a unified view that is then considered as a geospatial time series data representing the relationships between meteorology and vehicle (and probably city dwellers) mobility conditions. Such combination paves the way for answering advanced geospatial over time series queries that would be otherwise intractable. It is envisaged to operate as it follows: we will build an end-to-end pipeline that can exploit a tweaked version of the efficient

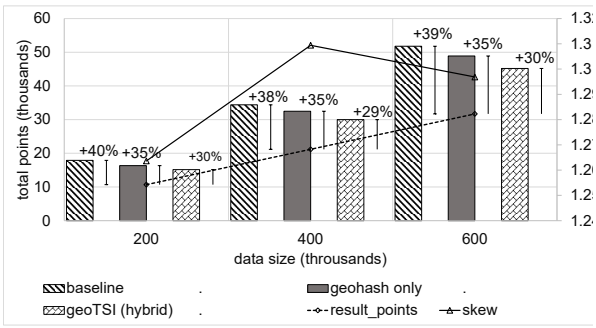

Fig. 5 total points, baseline against geohash and geoTSI

integration method that we have designed in our previous work [11] to integrate mobility and meteorological data on temporal and locational dimensions, at scale. Thereafter, a consequent stage in the pipeline will contain the multilevel indexing method that we have presented in the current paper, to enable efficient geospatial over time series queries robustly on the combined data. The two stages will be operating synergistically to achieve this purpose. This is useful in detecting the relationships between temporal and spatial dimensions of the combined data, thus simplifying the prediction of future events. For example: "how the speeds of vehicles in various segments of streets in a city are related to the temporal fluctuations in the values of pollution indexes such as Particulate Matters (PM10 and PM2.5)". This, in turns, can be used for informing a realistic decision regarding restructuring and redesigning parts of a city, taking into considerations the pollution levels and how to reduce those levels to protect the health of inhabitants and city dwellers.

Also, we will focus on developing a novel sampling method for fast-arriving georeferenced time series big data. We are planning to achieve this by extending a well-performing method that we have designed previously for sampling geospatial big data streams [12, 13]. Thereafter, we plan to enforce mechanisms that renders the TSDB aware of the quality-of-service (QoS) by enabling approximate query processing on georeferenced time series big data, similar to the way we have done for the geospatial counterpart in a previous work [14].

Our method described in this paper is however not a panacea and it has some limitations. The partial set of series data that results from applying the geohash filter need to be redistributed in accordance with a second key in the subsequent filtering stage (i.e., the S2 index), which takes tiny toll despite being amortized by the performance benefits of the overall method.

ACKNOWLEDGMENT

This research was supported by the project "H2020SIMDOME – Digital Ontology-based Modelling Environment for Simulation of Materials". We also would like to thank Microsoft for providing us with the free Microsoft Azure resources (through the AI for Earth project) through our project titled "Supporting Highly-Efficient Machine Learning Applications for Reducing the Impact of Climate Change on Human Health in Metropolitan Cities". All the tests for the results obtained in this paper have been conducted on an Azure deployment (as part of the foresaid project) that consists of an InfluxDB instance deployed on Azure.

REFERENCES

[1] K. Zoumpatianos, S. Idreos, and T. Palpanas, "Indexing for interactive exploration of big data series," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 1555-1566.

[2] A. Camerra, J. Shieh, T. Palpanas, T. Rakthanmanon, and E. Keogh, "Beyond one billion time series: indexing and mining very large time series collections with i sax2+," *Knowledge and information systems,* vol. 39, no. 1, pp. 123-151, 2014.

[3] I. M. Al Jawarneh, P. Bellavista, F. Casimiro, A. Corradi, and L. Foschini, "Cost-effective strategies for provisioning NoSQL storage services in support for industry 4.0," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, 2018: IEEE, pp. 01227-01232.

[4] I. M. Al Jawarneh, P. Bellavista, A. Corradi, L. Foschini, and R. Montanari, "Efficient QoS-Aware Spatial Join Processing for Scalable NoSQL Storage Frameworks," *IEEE Transactions on Network and Service Management,* 2020.

[5] I. M. Aljawarneh, P. Bellavista, A. Corradi, R. Montanari, L. Foschini, and A. Zanotti, "Efficient spark-based framework for big geospatial data query processing and analysis," in *2017 IEEE Symposium on Computers and Communications (ISCC)*, 2017: IEEE, pp. 851-856.

[6] I. M. Al Jawarneh, P. Bellavista, A. Corradi, L. Foschini, R. Montanari, and A. Zanotti, "In-memory spatial-aware framework for processing proximity-alike queries in big spatial data," in *2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2018: IEEE, pp. 1-6.

[7] M. Zaharia *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM,* vol. 59, no. 11, pp. 56-65, 2016.

[8] I. M. Al Jawarneh, P. Bellavista, A. Corradi, L. Foschini, and R. Montanari, "Locality-Preserving Spatial Partitioning for Geo Big Data Analytics in Main Memory Frameworks," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*, 2020: IEEE, pp. 1-6.

[9] G. Wang, X. Chen, F. Zhang, Y. Wang, and D. Zhang, "Experience: Understanding long-term evolving patterns of shared electric vehicle networks," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1-12.

[10] I. M. Al Jawarneh, P. Bellavista, A. Corradi, L. Foschini, and R. Montanari, "Big Spatial Data Management for the Internet of Things: A Survey," *Journal of Network and Systems Management,* vol. 28, no. 4, pp. 990-1035, 2020.

[11] I. M. Al Jawarneh, P. Bellavista, A. Corradi, L. Foschini, and R. Montanari, "Efficiently Integrating Mobility and Environment Data for Climate Change Analytics," in *2021 IEEE 26th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2021: IEEE, pp. 1-5.

[12] I. M. Al Jawarneh, P. Bellavista, A. Corradi, L. Foschini, and R. Montanari, "Spatially Representative Online Big Data Sampling for Smart Cities," in *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2020: IEEE, pp. 1-6.

[13] I. M. Al Jawarneh, P. Bellavista, L. Foschini, and R. Montanari, "Spatial-Aware Approximate Big Data Stream Processing," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019: IEEE, pp. 1-6.

[14] I. M. Al Jawarneh, P. Bellavista, A. Corradi, L. Foschini, and R. Montanari, "QoS-Aware Approximate Query Processing for Smart Cities Spatial Data Streams," *Sensors,* vol. 21, no. 12, 2021, doi: 10.3390/s21124160.