RASED: A Scalable Dashboard for Monitoring Road Network Updates in OSM

Mashaal Musleh, Mohamed F. Mokbel
Department of Computer Science and Engineering, University of Minnesota, MN, USA
{musle005, mokbel}@umn.edu

***Abstract*—Understanding the evolution and changes of digital road networks and how it resembles the true physical road network, have been a rich area of study within map analyzers, urban planners, and transportation communities. The main focus was to study OpenStreetMap (OSM) as the most commonly used platform for worldwide digital road networks, and is deemed even more accurate than commercial maps. However, all such studies have been localized to small areas of interest, mainly due to the large scale of the whole OSM road network. This paper presents RASED; a publicly available scalable dashboard to interactively monitor and analyze the evolution of all OSM road network. Using RASED, map analyzers can query and visualize various statistics about the road network daily changes worldwide, which would give a better understanding of the status of map quality and stability anywhere in the world. RASED relies on daily and monthly offline precomputations, accessed via a hierarchical temporal index structure. Experimental results show that RASED queries are always supported in the order of milliseconds, regardless of how large is the query temporal window, which allows highly interactive map analysis.**

I. Introduction

It used to be the case that accurate digital maps are only built and sold by major industry, e.g., HERE [12] and TomTom [15]. However, the high cost and proprietary nature of commercial maps along with their inherent inaccuracy due to not being able to be frequently updated, made researchers, developers, practitioners, and enterprises turn their attention towards open-source maps [5], [14], [28], [38]. A prime example of such maps is OpenStreetMap (OSM) [34], known as the Wikipedia of maps. OSM is a platform for crowdsourcing-based maps that has recently replaced commercial providers in various sectors of academia, government, and industry [29], [31]. For example, Amazon Logistics [1] is using OSM data in their delivery programs [33], Apple Maps [3] have been using OSM since iOS 6 [32], Facebook uses OSM as its backbone mapping support [10], Lyft has described OSM as the "Freshet Map for Rideshare" [24], while Tesla [39] uses OSM for its routing [40]. All these companies, and many others including Mapbox, Microsoft, and Uber, are not only using OSM, but are also extensively contributing to it [2], [9], [46].

Meanwhile, though there is extensive research in academia and industry for developing efficient algorithms for a myriad of road network queries (e.g., shortest path [21], [20], [44], [47], range [6], [19], [43], [49], and k-NN [4], [7], [17], [37], [45] queries), all algorithms have the implicit assumption that the underlying road network is accurate. Unfortunately, such an assumption is not always true as road networks suffer from all sorts of inaccuracy that significantly degrade the query result accuracy. While this may be acceptable for casual users where inaccuracy may only mean few minutes of delay, it is not the case for governmental or commercial applications that support map services for large numbers of users. For example, in USA, 99% of delivery company drivers say that they would be more efficient if they had better maps [26]. Problems, identified by those drivers, include: maps recommend longer routes and are not updated. This wastes significant time that translates into wages and high gas consumption, costing delivery companies $6B annually [41]. This trend is just going to increase with the increase of online shoppers and riders, which shifts traffic from casual users to delivery companies and ride sharing services.

Though OSM is deemed more accurate and up-to-date than commercial maps, its accuracy is still far from being acceptable for high-demand map services [11], [23], [25], [27], [42]. This has triggered several research efforts, mostly led by the transportation community, to study the quality of the underlying road network, represented by OSM (e.g., [13], [18], [48]). Unfortunately, all such quality assessment studies have very limited scope and scale, where the focus is only to study a certain city or country road network with heavy manual operations. Up to our knowledge, there is no comprehensive global-scale study for the quality of OSM road network. This is mainly due to its large scale, which makes researchers limit their studies to small regions. For example, OSM road network has more than 180M road segments and 2B nodes, which account for 500GB worth of raw data.

This paper introduces RASED (https://rased.cs.umn.edu); a publicly available scalable dashboard to interactively monitor and analyze all OSM road network updates worldwide. RASED is the first-ever attempt to quantify and visualize all OSM worldwide changes on a daily basis, which gives an idea about road network stability anywhere in the world. RASED provides the necessary infrastructure immensely needed by map analyzers to understand and assess the map quality. Using RASED, map analyzers can query and visualize various map statistics, including number and percentage of OSM updates per country, comparison between countries, types of updated roads, and temporal evolution of updates. All queries can have several filters including temporal (e.g., time of update), spatial (e.g., country or state), road types, and update types, which would all give a better understanding of map status globally.

This work is supported by the National Science Foundation, USA, under Grant IIS-1907855.

2375-0324/22/$31.00 ©2022 IEEE
DOI 10.1109/MDM55031.2022.00050

RASED is a highly interactive system, where all its analysis queries are supported in milliseconds allowing interactive visualization of the results. This makes RASED a convenient and highly important dashboard for road network map analyzers worldwide. To achieve its scalability and interactivity, the RASED backend employs: (1) *offline daily aggregation*, where the daily crawled OSM updates are analyzed offline to form all sorts of required precomputations, stored in data cubes [16], (2) *hierarchical indexing*, where the offline daily aggregation cubes form a hierarchical index of weekly and monthly updates to support analysis queries over longer time periods, and (3) *caching*, where some of the daily/weekly/monthly data cubes are prefetched in memory for faster access. All together, achieve a milliseconds response time when querying all OSM road network updates during the past 15 years.

The rest of this paper is organized as follows: Section II gives a brief background about OSM. Section III gives RASED system architecture. Section IV shows the queries supported by RASED. RASED three main modules, namely, Data Collection, Indexing, and Querying are described in Sections V, VI, and VII, respectively. Section VIII experimentally evaluates RASED. The paper is concluded in Section IX.

II. Background

OpenStreetMap (OSM) [34], launched in 2004, is a collaborative community project to create a free editable map of the world. Known as the Wikipedia of maps, OSM has 8.5 Million users, with 300K active users per year (users who made at least one edit during the year) [36]. OSM supports 400+ public free open-source OSM-based services [22], 80+ OSM-based commercial services [8], and receives API requests at the rate of 800 requests per second, for only one OSM data center [35]. This section gives a brief and necessary background about OSM data and update representation.

A. OSM Conceptual Data Model

OSM data is all stored in one big XML file (Plant.osm) presenting a massive list of elements, where each element is one of the following three types: (1) *Node*, which represents a certain point in the space with node identifier and its latitude and longitude coordinates. Objects represented by Nodes include intersection points, traffic lights, stop signs, bus stations, and other Points of Interest (PoI). (2) *Way*, which represents an ordered list of node identifiers making connected road segments. (3) *Relation*, which represents the relations between one or more elements of any type. Relations are used to model complex roads that may contain multiple parts (e.g., multiple Ways). Currently, OSM Planet.osm file is 1.6TB, and includes more than 7.5B nodes, 800M ways, and 9M relations.

B. OSM Map Updates

OSM is based on crowdsourcing where mappers voluntarily upload geographical data for their surroundings, which results in updating the map by creating new elements or modifying existing ones. OSM stores such updates in three different sets of files, described below:

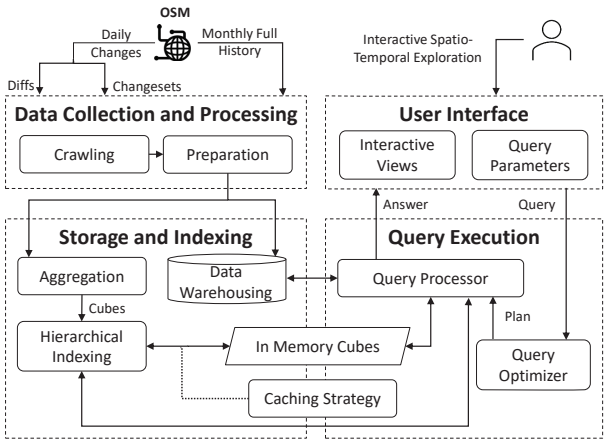

Fig. 1. RASED Architecture

Diff (https://wiki.openstreetmap.org/wiki/Planet.osm/diffs.) OSM creates such a file every minute, day, and hour such that any created or modified element is added (and replicated) to these three files. Only the element's after-image is stored in these files. Currently, OSM has 5M, 82K, and 3.5K minute, hourly, and daily Diff files, respectively, with sizes that range from a few megabytes to a few gigabytes per file.

Changesets (https://wiki.openstreetmap.org/wiki/Changeset.) A set of files that provide metadata information about map updates, e.g., user information, bounding box, comments, and sources, described for each changeset; a term used to represent all updates submitted by a particular user in one session (maximum of 24 hours). OSM provides two sources to download such data: (a) A single large file created every week with a dump of all changesets in OSM lifetime, currently of size 50GB. (2) A series of sequentially numbered small files (tens of kilobytes), such that a new file is created for every 1K new changesets. Currently, OSM approximately creates a new such file every minute and has created 5M files.

Full History (https://wiki.openstreetmap.org/wiki/Planet.osm/full.) One huge file dumped every few weeks for entire OSM updates. Unlike Diff files, the full history includes the previous state of each update. Currently, this file size is 3+TB and has 12+ Billion elements of all versions.

III. RASED Architecture

Figure 1 depicts the architecture of RASED, composed of the following four main modules:

User Interface. This module presents the Web Graphical User Interface (GUI) for RASED. It receives a set of interactive online queries from RASED users and sends it to the *Query Execution* module, which responds back with the answer in an interactive way. The query result is then visualized in various ways that allow map analyzers and domain experts to assess OSM stability and changes anywhere in the world. We will not discuss this module further in this paper. Interested readers can refer to the live RASED system and interact with it to explore

its friendly user interface at https://rased.cs.umn.edu and/or refer to RASED published demo [30] for detailed screenshots and description of RASED user interface.

Data Collection and Processing. This module is responsible for daily and monthly crawling of the OSM updates and preparing them for consumption by the *Storage and Indexing* module. The output of this module is a long list of daily/monthly updates, termed *UpdateList*, where each update has eight attributes: <*ElementType, Date, Country, Latitude, Longitude, RoadType, UpdateType, ChangesetID*>. *ElementType* is the type of the updated element (i.e., node, way, relation), *Date*, *Country*, *Latitude*, *Longitude* represent the date and location of the update, *RoadType* is the type of the updated road (e.g., highway, service, residential), *UpdateType* is the type of the update (e.g., new road, update geometry, deletion), *ChangesetID* is a reference to the changeset (Section II-B) that contains this update. Details are in Section V.

Storage and Indexing. This module takes the output of the *Data Collection* module as its input. Then it goes through two main operations: (a) computing various sorts of precomputations in a form of data cubes [16] and using it to populate its own hierarchical temporal index structure of daily, weekly, monthly, and yearly precomputed statistics, and (b) dumping the input to a traditional data warehouse indexed by both *ChangeSetID* and a spatial index. Details are in Section VI.

Query Execution. This module receives RASED queries submitted through the *User Interface* module, and answers them in an interactive way. Internally, it employs two main ideas: (a) *Caching*, where a selected set of aggregate data cubes are cached in memory to efficiently support incoming queries, and (b) *Level optimization*, where it smartly decides which level(s) in the index hierarchy would be better exploited for more efficient query support. Details are in Section VII.

IV. RASED Queries

This section describes the various queries supported by RASED, presented in a SQL format. The most important queries fall under the category of analysis queries, described in Section IV-A. Update sample queries are described in Section IV-B.

A. Analysis Queries

RASED analysis queries aim to provide detailed statistics about road network updates. Examples of such queries include: "*finding the number or percentage of road network updates over the last two years for a particular set of countries*", "*finding the number of updates for each road type for a certain country over a certain time period*", and "*compare the road network evolution for a particular set of countries*". The results of RASED analysis queries can be presented as either absolute numbers or percentages of the country's road network size, and can be visualized as: (a) tabular format sorted on any column, (b) various charts (bar, choropleth, time series), or (c) a timelapse video showing the road network evolution.

Generally speaking, RASED analysis queries are aggregate queries on a subset of the fields from the *UpdateList* relation, namely, *ElementType*, *Date*, *Country*, *RoadType*, and *UpdateType*, described in Section III. In particular, RASED queries would have the following SQL signature:

```
SELECT
  U.ElementType, U.Date, U.Country,
  U.RoadType, U.UpdateType, COUNT(*)
FROM UpdateList U
WHERE
  U.ElementType IN ListofElementTypes
  AND U.Date BETWEEN date1 AND date2
  AND U.Country IN ListofCountries
  AND U.RoadType IN ListofRoadTypes
  AND U.UpdateType IN ListofUpdateTypes
GROUP BY
  U.ElementType, U.Date,
  U.Country, U.RoadType, U.UpdateType
```

Below are few examples of RASED analysis queries and their visualized answer, based on the above query signature:

Example 1: Country Analysis. "*Find the number of newly created or modified element types (node, way, relation) for each country road network in 2021*": Out of the five attributes in the query signature, we would need to group on only two of them (*Country* and *ElementType*) as we need the answer for each country and each element type. We have conditions on both the *Date* and *UpdateType*. No group or constraints on the fifth attribute, *RoadType*.

```
SELECT U.Country,U.ElementType,COUNT(*)
FROM UpdateList U
WHERE U.Date BETWEEN 2021-01-01
  AND 2021-12-31
  AND U.UpdateType IN [New,Update]
GROUP BY U.Country, U.ElementType
```

Figures 2 and 3 give RASED visualization for that query in both bar chart and table formats, respectively.

Example 2: Road Type Analysis. "*Find the number of newly created or modified elements types (node, way, relation) for each road type in USA since 2018*": We group on two attributes (*RoadType* and *ElementType*) and have filters on the remaining three attributes, *Date*, *Country*, and *UpdateType*.

```
SELECT U.RoadType,U.ElementType,COUNT(*)
FROM UpdateList U
WHERE U.Date AFTER 2018-01-01
  AND U.Country = USA
  AND U.UpdateType IN [New,Update]
GROUP BY U.RoadType, U.ElementType
```

Figure 4 gives RASED visualization for the query answer.

Example 3: Comparative Time-Series Analysis. "*Compare the percentage of daily changes in road network in Germany, Singapore, and Qatar over 2020 and 2021*". We group and have conditions on *Country* and *Date*. Nothing is needed for the remaining three attributes.

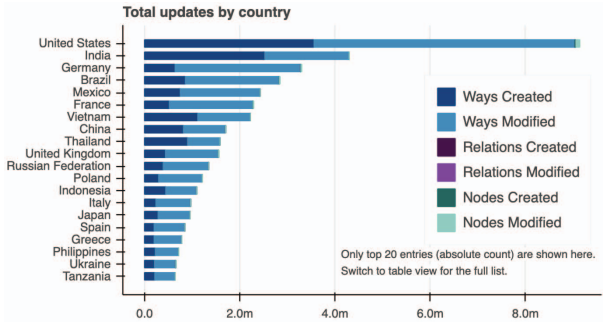

Fig. 2. Visualized Results for Country Analysis Example in Bar Chart Format

country	All	Ways Created	Ways Modified	Relations Cr	Relations Mc	Nodes Cr	Nodes Moc
United States	9,142,858	3,559,417.0	5,483,190.0	43.0	92.0	19,295.0	80,821.0
India	4,299,546	2,525,864.0	1,771,880.0	3.0	104.0	210.0	1,485.0
Germany	3,302,925	641,372.0	2,639,701.0	79.0	143.0	4,383.0	17,247.0
Brazil	2,847,160	857,852.0	1,971,562.0	4.0	9.0	4,068.0	13,665.0
Mexico	2,432,545	751,154.0	1,677,685.0	0.0	3.0	526.0	3,177.0
France	2,293,375	521,512.0	1,755,216.0	54.0	60.0	4,267.0	12,266.0
Vietnam	2,223,752	1,117,416.0	1,106,088.0	0.0	25.0	157.0	66.0

Fig. 3. Results for Country Analysis Example in Table Format

```
SELECT U.Country, U.Date, Percentage(*)
FROM UpdateList U
WHERE U.Date BETWEEN 2020-01-01
  AND 2021-12-31
  AND U.Country IN [Germany,
    Singapore, Qatar]
GROUP BY U.Country, U.Date
```

Figure 5 gives RASED visualization for the query answer.

B. Sample Update Queries

RASED users may want to see a sample of the updates that represent a given analysis query. Hence, RASED provides a query interface that visualizes a sample of N (default = 100) such updates on the map based on their *latitude* and *longitude* information. RASED also uses the *ChangesetID* of the samples to call a third-party application that visualizes the details of the sample update.

V. Data Collection and Processing

This module crawls OSM update files, described in Section II to produce the *UpdateList* of eight-attributes tuples: *<ElementType, Date, Country, Latitude, Longitude, RoadType, UpdateType, ChangesetID>*. One way to realize this module is to deploy a monthly crawler of OSM *full history* file. However, this would mean that RASED would have stale statistics, only updated on a monthly basis. Hence, we opt to have daily crawlers of the daily *diff* and *changesets* files, and use them to construct as much as possible of the *UpdateList*. Then, use the monthly crawler to complete the missing information. This ensures the accuracy of most RASED analysis queries.

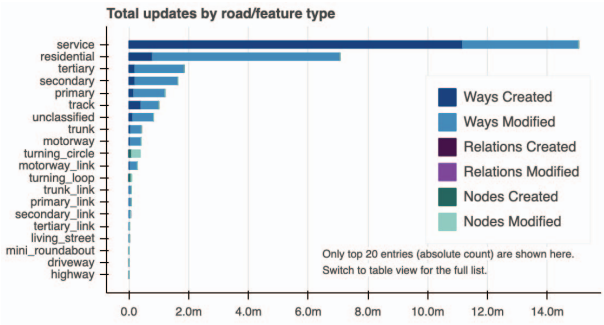

Fig. 4. Visualized Results for Road Type Analysis Example

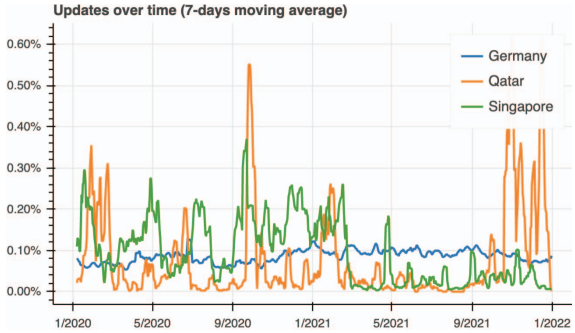

Fig. 5. Visualized Results for Comparative Time-Series Analysis Example

Daily Crawler. The daily crawler mainly constructs seven out of the eight attributes of the *UpdateList*. For the eighth attribute (*UpdateType*), we can only infer whether an update is a new or updated tuple, but would not know whether this is an update for geometry or for metadata. Hence, we would defer these details to the monthly crawler. For each update record, four out of the seven attributes (in addition to the *UpdateType*) are obtained in a straightforward way from the *diff* files, namely, *ElementType*, *Date*, *RoadType*, and *ChangesetID* attributes. The remaining three attributes, *Country*, *Latitude*, *Longitude*, can only be easily obtained for the *node* elements, but not for the *way* and *relation* elements. To find out such information for each update tuple, we use its *ChangesetID* to retrieve its bounding box from the *changesets* file. We then map the bounding box to its country, and assign latitude and longitude coordinates based on the center point contained in the bounding box.

Monthly Crawler. The monthly crawler is made to go through the *full history* file to compare every two consecutive versions of an element and classify the update type as either *create*, *delete*, *metadata update*, or *geometry update*. Newly created elements will always be their first version, while deleted ones are the last version. Geometry updates occur when there is a change in the latitude/longitude attributes or the list of members of a *way* or *relation* element, while metadata update occurs by changing the element tags.

VI. Storage and Indexing

The Storage and Indexing module takes the daily and monthly *UpdateList*: <*ElementType, Date, Country, Latitude, Longitude, RoadType, UpdateType, ChangesetID*>, produced from the Data Collection module (Section V) as its input. Then, it builds and maintains a storage infrastructure that can be efficiently accessed by the Query Execution module (Section VII) to support RASED queries. This section describes such storage infrastructure per the type of supported queries.

A. Supporting Analysis Queries

To support RASED analysis queries (Section IV-A), we build and maintain a hierarchical temporal index structure that ensures that all queries will be supported with very few I/Os. This gives an interactive user experience navigating through various analysis queries. We describe below the index hierarchy, index nodes, index size, and index maintenance.

Index Hierarchy. Figure 6 depicts the hierarchical temporal index structure, employed by RASED. The index does not index the OSM updates itself. Instead, it indexes precomputed statistics (i.e., aggregates) about the OSM updates. These precomputed statistics basically cover everything one could ask for from any RASED analysis query. The index has four levels that represent yearly, monthly, weekly, and daily statistics with one dummy root node at the top that points to the various yearly statistics. All statistics are presented in the form of data cubes [16], each is stored in one-page index node. Each yearly statistics is basically an aggregation of twelve monthly statistics. In turn, the monthly statistics are aggregates of four weekly and zero to three daily statistics, and so on.

Index Nodes. Each index node at any level is basically a four-dimensional data cube [16], where the dimensions correspond to four attributes from the *UpdateList*, namely, *ElementType*, *Country*, *RoadType*, and *UpdateType*. In RASED, we have the following possible values for each dimension: (1) *ElementType*. Three possible values, presenting node, way, and relation elements. (2) *Country*. 300+ values presenting all countries plus some selected zones of interest (e.g., continents and US states). (3) *RoadType*. 150 possible road types, including highway, residential, service, and truck roads. (4) *UpdateType*. Four kinds of update operations, namely, newly created roads/nodes, deleted roads/nodes, road geometry update, and road metadata update. This means that each cube maintains 540,000 precomputed values. Each cube cell is basically the count of OSM updates that happen in the time window of the cube (year, month, week, day) and match the corresponding value for each of the four dimensions.

Index Size. All nodes at all levels are of fixed size. With 540K values per node, each node takes ∼4MB of storage, which directly fits in one disk page. Considering all the OSM updates since its inception in 2004, we have 6,000+ daily nodes, 850+ weekly nodes, 200+ monthly nodes, and 16 yearly nodes. So, the total required storage is ∼28GB, which accommodate close to 7,000 nodes with 4 billion aggregate values. Though we store all index nodes on disk, the query executor caches some of them in memory for faster processing.

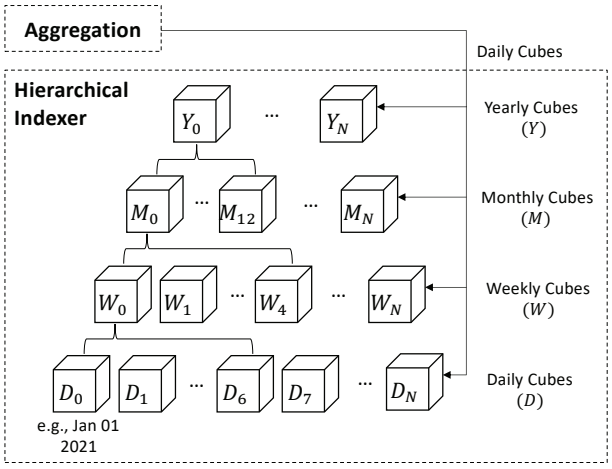

Fig. 6. Hierarchical Temporal Index for Data Cubes

Index Maintenance with Daily Updates. Once the daily *UpdateList* is received from the daily crawler process in Section V, we scan all the updates (10∼20MB), and construct a new data cube of 540,000 aggregate values as described above. Notice that with the daily crawlers, we would have only two possible values for the *UpdateType* dimension, so, in fact, we would calculate only 270,000 aggregate values, while putting zeros in the rest of the data cube cells. The cube is then stored in a newly allocated disk page and linked to last day cube. If this day is the end of the week, we construct the parent weekly cube by reading the six previous cubes and summing up their corresponding values to build a newly weekly cube. We do so recursively for monthly and yearly cubes, if this day is the end of the month and year, respectively. This process is performed offline, and takes up to 30 minutes. The time is mainly spent in scanning the *UpdateList*, and hence it depends on the number of updates of each day. The process does not consume much I/Os. Normally, we would need only one I/O for daily cubes. If it is the end of the week/month/year, we would need up to 8, 6, and 13 I/Os, respectively.

Index Maintenance with Monthly Updates. Once the monthly *UpdateList* is received from the monthly crawler process in Section V, we scan all the updates and reconstruct all the daily and weekly data cubes in that month. The main reason is that by now we have more detailed information about the *UpdateType* with four possible values. This would be a bit costly operation that would take a few hours due to the size of the monthly *UpdateList* and the number of I/O operations. Yet, the process is completely done offline, and copied to the index structure only when done.

B. Supporting Sample Update Queries

To support sample update queries (Section IV-B), we dump the whole *UpdateList* into a standard database table indexed by: (a) a hash index on *ChangesetID*, which is needed to retrieve a single update for RASED users to see the change that took place for a specific object, and (b) a spatial index on <*Latitude, Longitude*>, which is needed to retrieve the sample updates located in a certain spatial region.

VII. Query Execution

The Query Execution module supports RASED queries through efficient data retrieval from the index infrastructure laid out by the Storage and Indexing module. This section only focuses on supporting RASED analysis queries (Section IV-A), as sample update queries are supported through a straightforward index-based retrieval from a traditional DBMS. RASED query execution goes through two main phases: The first phase is mostly disk-based as it retrieves the data cubes that include the answer for a given query. The second phase is completely in-memory, where some computations may still be needed to aggregate values within the cube. For example, a query that asks about the number of updates in each country in a certain time window t, would first retrieve the data cubes that satisfy t. Since each cell in each cube represents one value of the four dimensions, *Country*, *ElementType*, *RoadType*, and *UpdateType*), we would then need to aggregate the values across three dimensions as we are only interested in the sum of updates for each country. The first phase is actually the bottleneck of this module, as the second phase is executed all in-memory. To reduce the overhead of the first phase, we employ two optimization techniques, *caching* (Section VII-A) and *level optimization* (section VII-B), geared towards reducing the number of retrieved data cubes from disk.

A. Caching Strategy

The idea of caching is to preload into memory some of the very recent data cubes, such that queries over recent data would be either fully or partially answered from in-memory cubes. This would significantly save from the query response time as we reduce the number of disk retrieval of data cubes. The rationale is that RASED is more likely to receive inquiries about recent updates than older ones. The challenge is from which index level we should pick our preloaded data cubes. Hence, we formulate our caching strategy as follows: Given N available memory slots and the sets Y, M, W, D of yearly, monthly, weekly, and daily cubes, we preload the following cubes into memory: $\{D_{|D|-i}\}_{i=0}^{\alpha N} \cup \{W_{|W|-i}\}_{i=0}^{\beta N} \cup \{M_{|M|-i}\}_{i=0}^{\gamma N} \cup \{Y_{|Y|-i}\}_{i=0}^{\theta N}$ where α, β, γ, and θ has a total sum of 1 and present the ratio of the N memory slots that will be allocated to each daily, monthly, weekly, and yearly levels, respectively. Such parameters present a trade-off between aggregation granularity and time coverage. For example, higher α would cache more daily details but less covered period, while higher γ and θ would favor longer period queries.

B. Level Optimization

A given analysis query could be answered from a mix of data cubes at different temporal levels. For example, an aggregate query for the period Jan 1, 2022 to Feb 15, 2022 can be answered using either: (a) 46 daily cubes, (b) six weekly cubes (weeks of Jan 2, 9, 16, 23, 30, and Feb 6) and four daily cubes (Jan 1 and Feb 13-15), or (c) one monthly cube (January), one weekly cube (week of Feb 6), and eight daily cubes (Feb 1 to 5 and 13-15). The objective of the level optimization is to find the query plan that would retrieve from disk the least number of data cubes, taking into consideration that some of the data cubes are already in memory due to the deployed caching strategy. For example, trying to reduce the number of data cubes in the previous example would directly advise using either plan (b) or plan (c) as both plans would only require 10 data cubes. However, if the caching strategy is set with a high value of α, then it could be that the last 60 daily cubes are in memory, and none of the other higher temporal level cubes. Hence, plan (a) would be favored here as it has zero disk access, while plans (b) and (c) would require six and two disk cubes, respectively.

VIII. Experiment

This section provides experimental evaluation of RASED to: (a) setup RASED parameters in terms of cache size and number of index levels (Section VIII-A), (b) understand the performance gain from employing caching and level optimization strategies (Section VIII-B), and (c) compare RASED overall performance against a traditional DBMS implementation (Section VIII-C). All experiments are done on an actual deployment of RASED as a publicly available web service at: https://rased.cs.umn.edu. For evaluation, we use the OSM full history dump which contains more than 12 billion updates with a total size of 3 TB of raw data. We run the whole dataset through RASED Data Collection module to come up with the full *UpdateList* as: *<ElementType, Date, Country, Latitude, Longitude, RoadType, UpdateType, ChangesetID>*, then bulk load the list into RASED temporal hierarchical index structure. We focus our experiments only on analysis queries (Section IV-A), as sample update queries (Section IV-B) are executed in a traditional DBMS way, so, there is nothing much to report about it. Our main performance measure is the query response time, which needs to be in order of milliseconds to ensure an interactive user experience of RASED analysis queries. Each point reported in all performance experiments is an average of 100 query execution. Unless mentioned otherwise, each query retrieves only one data cube cell to focus our performance results on the disk retrieval time, the default cache size N is 2GB, with α, β, γ, and θ are set to 0.4, 0.35, 0.2, and 0.05 respectively. All experiments are done using an Ubuntu system running on 8-core Intel(R) i7-4790 CPU @ 3.60GHz and 32GB of memory.

A. Setting RASED Parameters

This experiment aims to set the parameters of RASED index structure, namely the cache size and the number of levels in the hierarchical index. Figure 7 gives the query response time of RASED when varying the cache size from 128MB to 4GB, which can fit from 32 to 1,000 data cubes. We perform this experiment using various query loads with a time span of 1, 3, 6, and 12 months, which would reflect on the number of data cubes needed to answer each query. Clearly, the larger the cache size, the better the performance as higher numbers of data cubes can be retrieved from memory. For each query temporal window, there is a saturation point where increasing

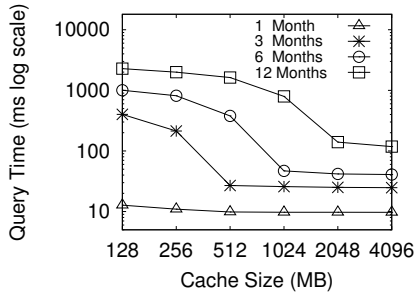

Fig. 7. Setting RASED cache size

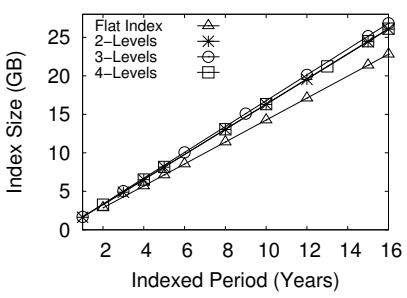

Fig. 8. Setting RASED number of levels

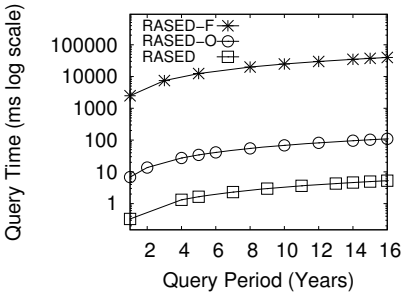

Fig. 9. Effect of Each Component in RASED

the cache size will not have significant enhancement, e.g., 512MB, 1024MB, and 2048MB for the queries with 3, 6, and 12 months, respectively. Since RASED supports queries with large time windows, we opt to choose 2048MB cache size in RASED deployment. Figure 8 gives the size needed for each additional hierarchy level for RASED index when varying the covered period from one to 16 years, where a flat index means one level of daily cubes, while extra levels are for weekly, monthly, and yearly cubes. Apparently, the extra levels do not add much beyond the storage already needed for the first daily level. In particular, a four-levels index for a 16-years period would only take 1.15 of storage taken by a flat index for the same period. Hence, we opt to have our hierarchical index with four levels.

B. RASED Query Execution Strategies

This section aims to understand the performance gain from employing caching and level optimization strategies in RASED. In particular, Figure 9 gives the performance of three variants of RASED when varying the query time window from one to 16 years. The first variant (RASED-F) is a one-level flat index with neither caching nor level optimization. The second variant (RASED-O) is the full RASED index with level optimization, but no caching. The third variant is the full RASED system with both level optimization and caching. The more than two orders of magnitude performance gain from RASED-F to RASED-O shows the impact of having the index hierarchy, along with the level optimizer. Meanwhile, the order of magnitude performance gain from RASED-O to RASED shows the impact of deploying the caching strategy. Overall, both index hierarchy and caching boost RASED performance by three orders of magnitude.

C. Overall Performance

This section evaluates RASED against PostgreSQL implementation of the RASED analysis queries. To ensure fairness, we set PostgreSQL buffer size to 2GB similar to RASED cache size. Figure 10 gives the performance of RASED and PostgreSQL when varying the query time window from one to 16 years. PostgreSQL constantly takes around 1000 seconds to answer the analysis queries regardless of the query period or

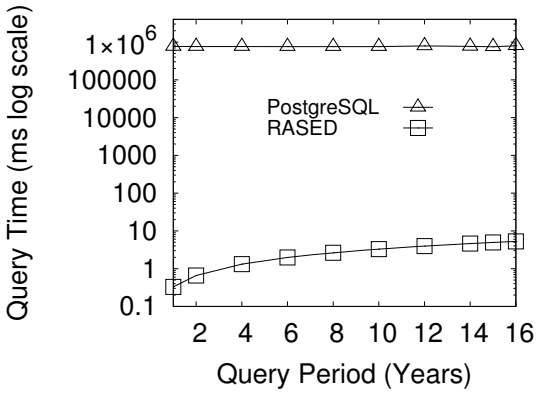

Fig. 10. Effect of Each Component in RASED

the aggregation size. This is mainly because it requires scanning the whole data since the query involves multiple attributes in the *Group By* clause. Meanwhile, RASED consistently achieves five to six orders of magnitudes better performance, reaching up to 10 milliseconds in its longest query period, which is due to its powerful index structure.

IX. Conclusion

This paper presented RASED; a publicly available scalable dashboard to interactively monitor and analyze all OpenStreetMap (OSM) road network daily updates worldwide. RASED supports a myriad of analysis queries that provide detailed statistics about road network daily updates activity, e.g., finding the number or percentage of road network updates over the last two years for a particular set of countries, finding the number of updates for each road type for a certain country over a certain time period, and comparing the road network evolution for a particular set of countries. RASED is equipped with a hierarchical temporal index structure and caching strategy that efficiently retrieve precomputed statistics needed for analysis queries. Results of RASED queries are visualized as either tabular format, various charts, or a timelapse video. RASED is highly interactive with milliseconds response to all its analysis queries. Realization of RASED has orders of magnitudes better performance than realizing similar ideas using traditional PostgreSQL DBMS.

REFERENCES

[1] Amazon Delivery and Logistics. https://www.aboutamazon.com/what-we-do/delivery-logistics.

[2] Jennings Anderson, Dipto Sarkar, and Leysia Palen. Corporate Editors in the Evolving Landscape of OpenStreetMap. *ISPRS International Journal of Geo-Information*, 8(5):232, 2019.

[3] Apple Maps. https://www.apple.com/maps/.

[4] Jie Bao, Chi-Yin Chow, Mohamed F. Mokbel, and Wei-Shinn Ku. Efficient Evaluation Of K-Range Nearest Neighbor Queries In Road Networks. In *MDM*, 2010.

[5] Bloomberg CityLab. Who Owns the Digital Map of the World? www.bloomberg.com/news/articles/2015-06-25/mapbox-openstreetmap-and-the-future-of-the-global-digital-mapping-industry.

[6] Ling Chen, Yanlin Tang, Mingqi Lv, and Gencai Chen. Partition-Based Range Query For Uncertain Trajectories In Road Networks. *GeoInformatica*, 19(1):61–84, 2015.

[7] Mirla Rafaela Rafael Braga Chucre, Samara Martins do Nascimento, José Antônio Fernandes de Macêdo, José Maria Monteiro, and Marco Antonio Casanova. Taxi, Please! A Nearest Neighbor Query In Time-Dependent Road Networks. In *MDM*, 2016.

[8] Commercial OSM Software and Services. https://wiki.openstreetmap.org/wiki/Commercial_OSM_Software_and_Services.

[9] Corey Dickinson. Inside the Wikipedia of Maps Tensions Grow Over Corporate Influence. Bloomberg. https://www.bloomberg.com/news/articles/2021-02-19/openstreetmap-charts-a-controversial-new-direction.

[10] Facebook AI. Mapping roads through deep learning and weakly supervised training. https://ai.facebook.com/blog/mapping-roads-through-deep-learning-and-weakly-supervised-training/.

[11] Facebook Engineering. MaRS: How Facebook keeps maps current and accurate. https://engineering.fb.com/2019/09/30/ml-applications/mars/.

[12] Geo Awesomeness. What does the acquisition of HERE mean for Nokia, carmakers, TomTom, Google and the industry? https://geoawesomeness.com/what-does-the-acquisition-of-here-mean-for-nokia-carmakers-tomtom-google-and-the-industry/.

[13] Jean-Francois Girres and Guillaume Touya. Quality Assessment of the French OpenStreetMap Dataset. *Transactions in GIS*, 14(4):435–459, 2010.

[14] GIS Lounge. Businesses Using Open Source GIS. https://www.gislounge.com/businesses-using-open-source-gis/.

[15] GPS World. TomTom-Tele Atlas Merger a Done Deal. https://www.gpsworld.com/consumer-oemnewstomtom-tele-atlas-merger-a-done-deal-2911/.

[16] Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total. In *ICDE*, 1996.

[17] Ling Hu, Yinan Jing, Wei-Shinn Ku, and Cyrus Shahabi. Enforcing K Nearest Neighbor Query Integrity On Road Networks. In *SIGSPATIAL*, 2012.

[18] Kent T. Jacobs and Scott W. Mitchell. OpenStreetMap Quality Assessment using Unsupervised Machine Learning Methods. *Transactions in GIS*, 24(5):1280–1298, 2020.

[19] Hoyoung Jeung, Man Lung Yiu, Xiaofang Zhou, and Christian S. Jensen. Path Prediction And Predictive Range Querying In Road Network Databases. *VLDB J.*, 19(4):585–602, 2010.

[20] Lei Li, Mengxuan Zhang, Wen Hua, and Xiaofang Zhou. Fast Query Decomposition for Batch Shortest Path Processing in Road Networks. In *ICDE*, 2020.

[21] Lingxiao Li, Muhammad Aamir Cheema, Mohammed Eunus Ali, Hua Lu, and David Taniar. Continuously Monitoring Alternative Shortest Paths on Road Networks. *PVLDB*, 13(11):2243–2255, 2020.

[22] List of OSM-based services. https://wiki.openstreetmap.org/wiki/List_of_OSM-based_services.

[23] Lyft Engineering. How Lyft Creates Hyper-Accurate Maps from Open-Source Maps and Real-Time Data. https://eng.lyft.com/how-lyft-creates-hyper-accurate-maps-from-open-source-maps-and-real-time-data-8dcf9abdd46a.

[24] Lyft Engineering. How Lyft discovered OpenStreetMap is the Freshest Map for Rideshare. https://eng.lyft.com/how-lyft-discovered-openstreetmap-is-the-freshest-map-for-rideshare-a7a41bf92ec.

[25] Lyft mapping team. ground truth evaluation of openstreetmap quality in north american cities. https://drive.google.com/file/d/1SbdOUjeP1Ljqz4ra931D3Pe8B5C3pde/view.

[26] Mapillary. Unveiling the Mapping in Logistics Report: The Impact of Broken Maps on Last-Mile Deliveries. https://blog.mapillary.com/update/2020/02/14/mapping-in-logistics.html.

[27] Marco Minghini and Francesco Frassinelli. OpenStreetMap History for Intrinsic Quality Assessment: Is OSM up-to-date? *Open Deospatial Data, Software, and Standards*, 4(9):1–17, 2019.

[28] Money Control News. Uber may shun Google Maps for open source ones: Report. www.moneycontrol.com/news/business/uber-may-shun-google-maps-for-open-source-ones-report-2764111.html.

[29] Joe Morrison. OpenStreetMap is Having a Moment: The Billion Dollar Dataset Next Door. Medium Artcile. https://joemorrison.medium.com/openstreetmap-is-having-a-moment-dcc7eef1bb01.

[30] Mashaal Musleh and Mohamed F Mokbel. A Demonstration of RASED: A Scalable Dashboard for Monitoring Road Network Updates in OSM. In *ICDE*, 2022.

[31] Nextbillion.ai. OpenStreetMap for Businesses: A Primer. White Paper. https://nextbillion.ai/whitepapers/OpenStreetMap-for-Businesses-A-Primer.

[32] OpenStreetMap Blog. Apple Maps. https://blog.openstreetmap.org/2012/10/02/apple-maps/.

[33] OpenStreetMap Wiki. Organised Editing/Activities/Amazon. https://wiki.openstreetmap.org/wiki/Organised_Editing/Activities/Amazon.

[34] OpenStreetMap. http://www.openstreetmap.org/.

[35] OSM API Calls Dashbord per Server (Culebre). https://prometheus.openstreetmap.org/d/9xY_21OMk/apache?orgId=1&refresh=1m&var-instance=culebre&from=now-7d&to=now.

[36] OSM Statistics. https://wiki.openstreetmap.org/wiki/Stats.

[37] Dian Ouyang, Dong Wen, Lu Qin, Lijun Chang, Ying Zhang, and Xuemin Lin. Progressive Top-K Nearest Neighbors Search in Large Road Networks. In *SIGMOD*, 2020.

[38] Studio Software Blog. Google Maps vs OpenStreetMap: Which One Is Better for Your Project? https://studiosoftware.com/blog/google-maps-vs-openstreetmap/.

[39] Tesla. https://www.tesla.com/.

[40] Tesmanian. Tesla and SpaceX News. Tesla Owners Improve Smart Summon Routes by Updating Open Street Maps. https://www.tesmanian.com/blogs/tesmanian-blog/tesla-owners-smart-summon-routes-open-street-maps-full-self-driving.

[41] Traffic Technology Today. Poor maps costing delivery companies US $6bn annually. https://www.traffictechnologytoday.com/news/mapping/poor-maps-costing-delivery-companies-us6bn-annually.html.

[42] Uber engineering. enhancing the quality of uber maps with metrics computation. https://eng.uber.com/maps-metrics-computation/.

[43] Haojun Wang and Roger Zimmermann. Processing of Continuous Location-Based Range Queries on Moving Objects in Road Networks. *TKDE*, 23(7):1065–1078, 2011.

[44] Lingkun Wu, Xiaokui Xiao, Dingxiong Deng, Gao Cong, Andy Diwen Zhu, and Shuigeng Zhou. Shortest Path and Distance Queries on Road Networks: An Experimental Evaluation. *PVLDB*, 5(5):406–417, 2012.

[45] Bin Yao, Zhongpu Chen, Xiaofeng Gao, Shuo Shang, Shuai Ma, and Minyi Guo. Flexible Aggregate Nearest Neighbor Queries In Road Networks. In *ICDE*, 2018.

[46] Deborah Yates. How Facebook, Apple and Microsoft are contributing to an openly licensed map of the world. The Open Data Institute (ODI). https://theodi.org/article/how-are-facebook-apple-and-microsoft-contributing-to-openstreetmap/.

[47] Ziqiang Yu, Xiaohui Yu, Nick Koudas, Yang Liu, Yifan Li, Yueting Chen, and Dingyu Yang. Distributed Processing of k Shortest Path Queries over Dynamic Road Networks. In *SIGMOD*, 2020.

[48] Hongyu Zhang and Jacek Malczewski. Accuracy Evaluation of the Canadian OpenStreetMap Road Networks. *International Journal of Geospatial and Environmental Research*, 5(2):1:1–1:14, 2018.

[49] Kai Zheng, Goce Trajcevski, Xiaofang Zhou, and Peter Scheuermann. Probabilistic Range Queries For Uncertain Trajectories On Road Networks. In *EDBT*, 2011.