TAREEG: A MapReduce-Based Web Service for Extracting Spatial Data from OpenStreetMap*

Louai Alarabi, Ahmed Eldawy, Rami Alghamdi, Mohamed F. Mokbel

KACST GIS Technology Innovation Center, Umm Al-Qura University, Makkah, KSA
{larabi,eldawy,rghamdi,mmokbel}@gistic.org
University of Minnesota, Minneapolis, MN 55455, USA
{louai,eldawy,alghamdi,mokbel}@cs.umn.edu

ABSTRACT

Real spatial data, e.g., detailed road networks, rivers, buildings, parks, are not really available in most of the world. This hinders the practicality of many research ideas that need a real spatial data for testing experiments. Such data is often available for governmental use, or at major software companies, but it is prohibitively expensive to build or buy for academia or individual researchers. This demo presents TAREEG; a web-service that makes real spatial data, from anywhere in the world, available at the fingertips of every researcher or individual. TAREEG gets all its data by leveraging the richness of OpenStreetMap dataset; the most comprehensive available spatial data of the world. Yet, it is still challenging to obtain OpenStreetMap data due to the size limitations, special data format, and the noisy nature of spatial data. TAREEG employs MapReduce-based techniques to make it efficient and easy to extract OpenStreetMap data in a standard form with minimal effort. TAREEG is accessible via `http://www.tareeg.org/`

1. INTRODUCTION

Taking advantage from the explosion in the number of GPS-enabled smart devices and the ubiquity of location services, there has been an increasing interest from industry and academia to advance the state of the art in location-based services that include shortest path queries, k-nearest-neighbor queries, range queries, spatial keyword search, and others (e.g., [9, 4, 10]). However, a major obstacle in progressing this kind of research is the difficulty in getting hands on real spatial data. While this may not be a problem for major industries, who can afford buying actual and accurate spatial data from specialized companies (e.g., NavTeq), this is prohibitively expensive for small startups and academia.

*This work is supported in part by KACST GIS Technology Innovation Center at Umm Al-Qura University, under project GISTIC-13-14, and was done while the first, second, and fourth authors were visiting the center.

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.
Copyright 2014 ACM 978-1-4503-2376-5/14/06 ...$15.00.
http://dx.doi.org/10.1145/2588555.2594528.

To fill in this gap, OpenStreetMap [6] has been launched in 2004 to allow volunteers to combine their efforts in building an exhaustive and trustworthy map for the whole planet with an increased focus given to road networks. OpenStreetMap whole world dataset is free and accessible as a file called *Planet.osm* in an XML format that is updated on daily basis, where its current size is 435 GB. Despite its richness, using this data is not an easy task due to its huge size and non-standard format. For example, to extract the road network of Istanbul, Turkey, one needs to parse the whole 435 GB file, and extract the parts that are related to Istanbul. Furthermore, in these parts, one needs to parse records carefully, to extract the XML tags that are related to road networks and exclude everything else. Also, we need to overcome the noisy data as many of the data there are contributed from volunteers. As a result, several attempts were proposed to extract data from OpenStreetMaps. For example, the GeoFabrik project [3] allows users to download a predefined area from OpenStreetMaps, yet it does not provide any kind of extraction of specific map features (e.g., road network). It is basically a range query service over existing data without any attempt to read the noisy format. Also, OSMOSIS [8] and OSM2PGSQL [7] are two proposed tools that attempt to process the whole *Planet.osm* by loading it on spatial databases. However these tools may take up to several days for loading the entire *Planet.osm* file. For instance, in [5], the authors report that they spent 305 hours (approximately 12 days) for loading and analyzing the data for their experiment. In another benchmark of OSM2PGSQL, importing the *Planet.osm* to database takes up to 7 days of processing [7].

In this demo, we present TAREEG; an easy and efficient web service tool to extract spatial data from OpenStreetMaps. TAREEG users can: (1) extract geographical information through a nicely designed web service with the daily updated geographical data on any arbitrary region on world map, (2) Export the extracted data in CSV, KML, and shapefile formats, and (3) Visualize the geographical data on the map using four different mapping engines, which are OpenStreetMap, Google Maps, Google Hybrid Maps, and Esri Maps. All TAREEG services are done very efficient and online. For most of the requests, it takes only few second to receive the extracted data via email.

The key idea behind the performance of TAREEG is that it uses the MapReduce paradigm to speed up data extraction, indexing, and querying from Planet.osm file. In particular, TAREEG is powered by SpatialHadoop [2]; an ex-

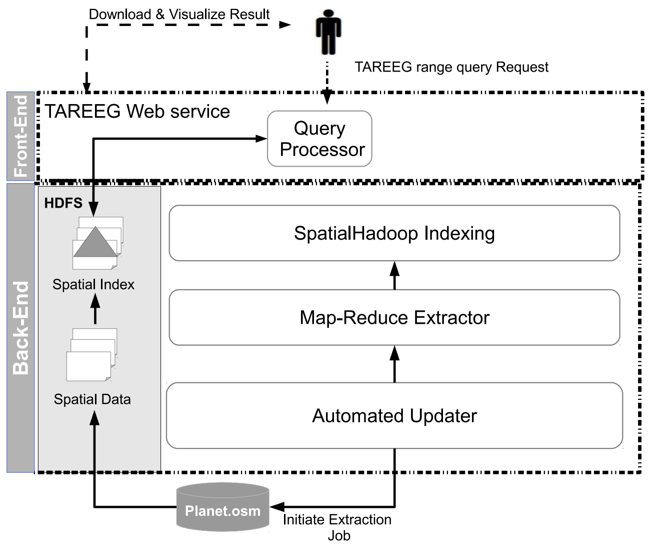

Figure 1: System Overview

tended MapReduce framework that deals efficiently with spatial data. In terms of data extraction and indexing, TAREEG takes advantage of SpatialHadoop and its distributed nature in the following: (1) each machine independently downloads part of the Planet.osm file that needs to be processed on that machine, decompress and process it on the fly. (2) TAREEG employs a special *file splitter* and *record reader* components that are capable of efficiently parsing the whole file and extracting its data in a distributed manner, and (3) TAREEG uses the indexing capabilities of SpatialHadoop to build efficient spatial indices for extracted data which allows user queries to run instantly on very large datasets. As a result, downloading and indexing the whole OpenStreetMap with all its spatial features takes few hours instead of few days. Then, querying this data takes few seconds instead of hours. The idea is that SpatialHadoop partitions this data over computing nodes using a spatial partitioning scheme, where spatially close by features are stored in the same file block.

2. SYSTEM OVERVIEW

Figure 1 gives the system overview of TAREEG. A map-reduce framework is used to handle a large scale dataset. A user interacts with TAREEG via its front-end that includes a web interface allowing users to submit their requests of any arbitrary region on the world map. Users may also visualize the extracted information via TAREEG front-end. Internally, TAREEG processes user requests by fetching the spatial information from spatial indexed files. TAREEG system back-end processes user requests through several processing stages: (1) Extraction stage, which is responsible for categorizing the contents of *Planet.osm* into bunch of homogeneous information stored separately, (2) Indexing stage, which is responsible for building spatial indexes on the extracted data, and (3) Querying stage, which is responsible on querying the indexed data.

As OpenStreetMap is updated on a daily or weekly basis, TAREEG employs an automatic updater module that is responsible for downloading the latest *Planet.osm* file and trigger the extraction and indexing stages. Automated module runs automatically daily once a new dataset is updated.

3. DATA EXTRACTION

Extracting information from OpenStreetMap is not a trivial task. OpenStreetMap dataset stores all data types sequentially in one large volume file in a semi-structured XML format. The XML file starts with nodes, then ways, and finally, relations. Tags are nested in each of these data types. The challenge of extracting information from Open StreetMap depends on identifying the annotation (i.e., tags) that imply categorized spatial data. We basically run a map-reduce job which takes the URL of the compressed Planet.osm file as an input. The output of data extraction will be several categorized files. Each file contains a homogeneous spatial information, e.g., road network data will be stored in one file and lakes in another one. In this phase, we implement a map-reduce program to extract information from the large scale OpenStreetMap data. We retrieve the URL of the latest version of *Planet.osm* and trigger a map-reduce job with the following three main components, namely splitter, Record Reader, and Mapper, described briefly below:

- **Splitter**: The *splitter* component breaks the input *Planet.osm* file into chunks of 64 MB (the default block size in HDFS). Since the file is compressed in *block zip* format, it is possible that each chunk is downloaded and decompressed separately to extract part of the XML file. Such splitting will cause inconsistent XML structures in each split which is handled later by the *Record Reader*. Thus, splitting the *Planet.osm* will parallelize and distribute the processing load into several map tasks, which is much faster than processing on a single node machine.

- **Record Reader**: Basically, default record reader in Hadoop processes text files line-by-line, so we can not process the data of *Planet.osm* inline. Due to the inconsistent representation of each split, we implemented an XML element reader instead of the default line reader provided by Hadoop framework. The output of this component is an XML *element* which will be sent later to the mapper. In addition, record reader is responsible for completing inconsistent elements in each split. To elaborate more, if a split has some missing information about one concise type such as *buildings*, the record reader will fetch these missing information from the next split. Then the element(s) are passed to the mapper.

- **Mapper**: This component receives *elements* sent by the Record Reader and classifies each element based on its annotation (i.e., tags), and then writes the result into separate files, based on the type of the extracted data, to HDFS, i.e., all road networks are written in one file, while lakes are written in another file, and so on. These data are stored randomly as a set of nodes N, which consists of $N = \{node\_id, longitude, latitude, tags\}$ and another set of extracted data R that consists of $R = \{edge\_id, node1\_id, node2\_id, tags\}$. R described as any type of geographical data, such as road network, lakes, buildings, or zip code, state, and national boundaries.

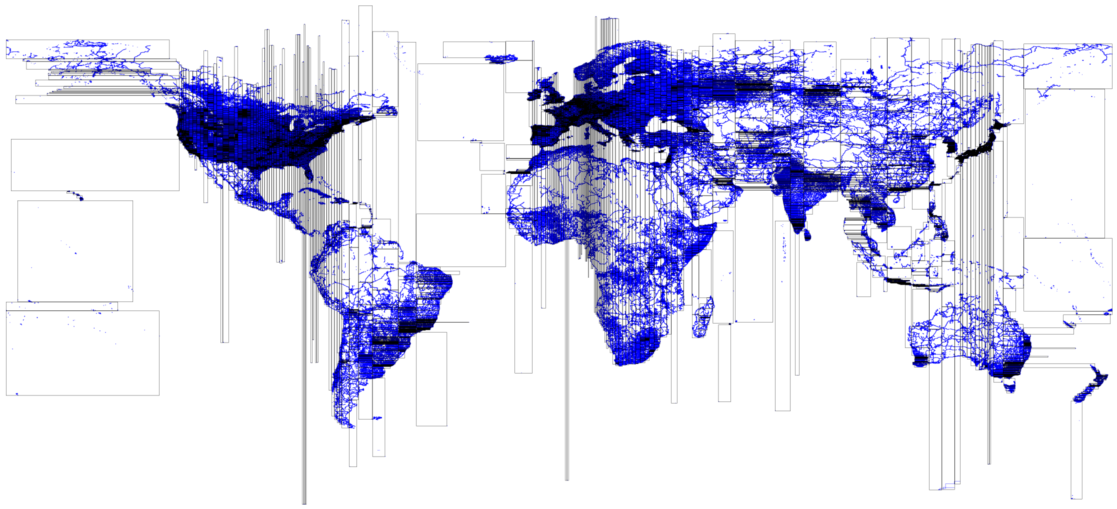

Figure 2: Spatial Data Partitioned

We use Pig [1] to combine the extracted data R with its spatial information (i.e., geolocation) from nodes N in a way similar to the join operation in any relational database. The output of the Pig program is a spatial set R' that consists of $R' = \{edge\_id, node1\_id, longitude1, latitude1, node2\_id, longitude2, latitude2, tags\}$. Each record $r' \in R'$ is now associated with the corresponding geo-location and neither sparsely nor spatially stored in HDFS.

4. INDEXING

Files generated by the extractor are heap files which are not organized in a specific order. This means that a range query executed to extract data in a specific region would have to scan the whole file to retrieve the result. This will considerably slow down the system, especially for larger datasets such as road network which has a total size of 130 GB. To speed up the processing of extracted datasets, we need to build spatial indexes for each dataset that efficiently supports range queries. One method to index datasets is to store each one as a relation in a spatial DBMS (e.g., PostGIS), and build a spatial index such as an R-tree on that relation. Once the relation is indexed, range queries can be expressed in SQL and executed efficiently inside the spatial DBMS. Unfortunately, this technique rendered infeasible due to the long time consumed to build the indexes.

To be able to construct the indexes efficiently in a timely manner, we use SpatialHadoop [2] to take advantage of its spatial indexing techniques. SpatialHadoop supports different types of spatial indexes including Grid file, R-tree, and R+-tree. Once datasets are extracted from the *Planet.osm* file, we use the SpatialHadoop '`index`' command on each file separately to build an index for it. The partition size is set to 64 MB (default for Hadoop). Each index is stored as one *master* file that stores the boundaries of each partition, and multiple data files with each file containing line segments overlapping with one partition.

Figure 2 gives the main reason behind the efficiency of spatial data indexing and querying in TAREEG. The figure shows an R+-tree partitioning scheme for the whole road network file. All road networks are depicted in blue lines, while the black rectangles in indicate partition boundaries. While some road segments cross multiple partitions, partition boundaries remain disjoint due to the properties of the R+-tree. As each partition is sufficient for only 64 MB worth of data, we can see that dense areas (e.g., Europe) are contained in very small partitions, while sparse areas (e.g., oceans) are contained in very large partitions. One way to look at this figure is that this is the way that SpatialHadoop divides a dataset of 455 GB into small chunks, each of 64 MB. Recall that in Hadoop, this 455 GB file will be divided into chunks of 64 MB as sequential heap file, which definitely does not fit spatial operations.

5. QUERY PROCESSING

Queries sent to TAREEG are basically range queries that request extracting a certain type of spatial data (e.g., road networks, lakes, buildings, borders) within a certain area of interest, presented as a rectangular area. When a range query is sent to TAREEG, it first executes that query on the master file to retrieve partitions that overlap the range query. For each matching partition, the records in the corresponding data file are compared to the query range and all matching records are stored as part of the answer. Since in R+-tree some records overlapping multiple partitions are replicated, a post processing duplicate avoidance step is executed to ensure the correctness of the answer as described in [2].

6. DEMONSTRATION SCENARIO

Conference audience can interact with TAREEG through two main functionalities: (1) *Data extraction request*, where audience can submit extraction requests through nicely designed web interface, by selecting any arbitrary region of world map and the type of spatial data they want to extract. (2) *Spatial data download and visualization*, where audience can download and visualize exported data on either OpenStreetMap, Google Maps, Google Hybrid Maps (i.e., hybrid satellite images with labels), and Esri Maps (i.e National Geographic map). A first version of TAREEG is currently accessible online at: `http://www.tareeg.org`

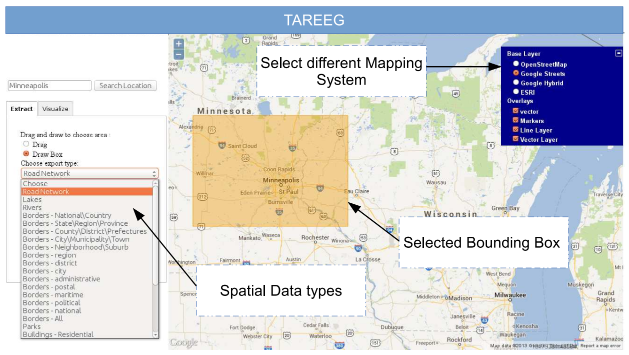

Figure 3: TAREEG Web Interface

6.1 Spatial Data Extraction Request

Initially, audience interact with TAREEG system by requesting different spatial data type as shown in Figure 3, following few steps: (1) the user selects any arbitrary region from the world map, either by zooming and draging the map to the geographical area of interest, or by searching the area of interest using the search bar, (2) the user specifies a boundary rectangle of the selected region, (3) the user provides an email address, in which TAREEG will send an email when the request is satisfied with links to download the requested data, and (4) the user submits the request by hitting the 'Extract button'.

Once a request is received, TAREEG processes this request by querying the requested spatial data from the back-end of the system, and notifies the requester through email once the data is available. It is important to note here that all requests to TAREEG are satisfied from its own local data, as there is no need to contact OpenStreetMap servers, and hence requests are usually satisfied within a few seconds. Once the requested data is available, the user will receive an email from TAREEG with hyperlinks to download the requested data. The time of processing the request depends on the size of the selected region and complexity of the requested dataset.

6.2 Spatial Data Download & Visualization

TAREEG users receive their extracted data in three common standard formats, a Comma Separated Values (CSV) files, Google Keyhole Markup Language (KML), and Esri geospatial vector (shapefile) formats . TAREEG only supports visualizing (CSV) format, where KML and shapefile can be already visualized through various applications (e.g QGIS, Google Earth). The CSV format has the following structure:

```
Node < Node_id , Latitude , Longitude >
Edge <Edge_id , Start_Node_id ,  End_Node_id ,  [Tags] >
```

Figures 4 and 5 give two examples of the visualization tool of TAREEG, where an extracted set of lakes and road networks are visualized, respectively. TAREEG visualizer shows the spatial data on the map by uploading the CSV files, that were sent to the user as hyperlinks in an email. TAREEG visualizer supports four different mapping engines, where audience can experience the easiness of switching the gear between these mapping engines.

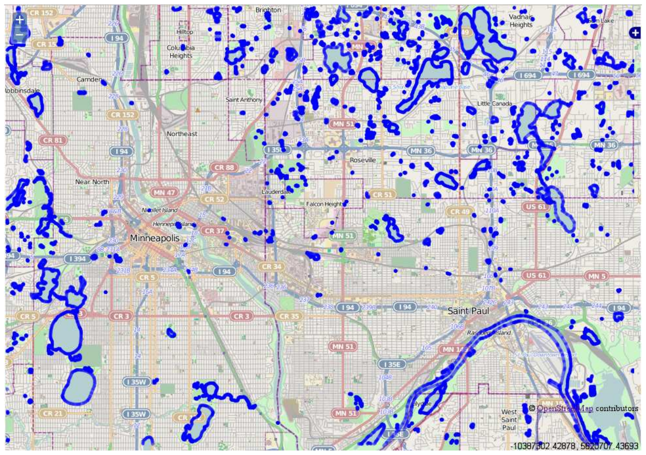

Figure 4: Extracted lakes

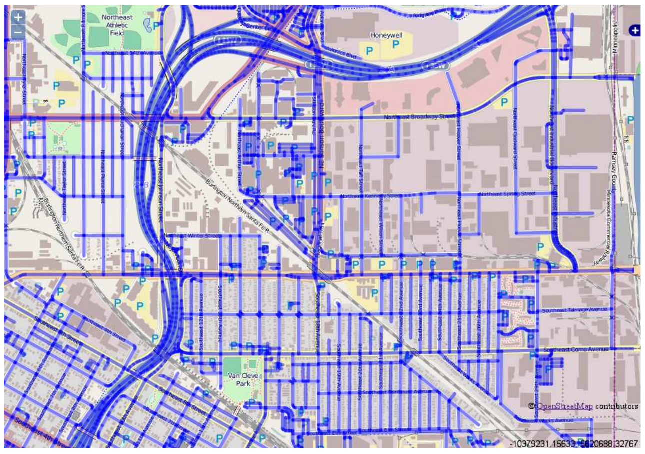

Figure 5: Extracted Road Networks

7. REFERENCES

[1] Apache. Pig http://pig.apache.org/, June 2013.
[2] A. Eldawy and M. F. Mokbel. A demonstration of spatialhadoop: An efficient mapreduce framework for spatial data. *PVLDB*, 2013.
[3] geofabrik. http://download.geofabrik.de/, Dec. 2013.
[4] S. Luo, Y. Luo, S. Zhou, G. Cong, and J. Guan. DISKs: a system for distributed spatial group keyword search on road networks. 5(12):1966–1969, 2012.
[5] P. Mooney and P. Corcoran. Characteristics of heavily edited objects in openstreetmap. *Future Internet*, 2012.
[6] OSM. http://www.openstreetmap.org, Dec. 2013.
[7] osm2pgsql. Benchmarks jun 2012 http://wiki.openstreetmap.org/wiki/Osm2pgsql/benchmarks.
[8] Osmosis. 2013 http://wiki.openstreetmap.org/wiki/Osmosis.
[9] J. R. Thomsen, M. L. Yiu, and C. S. Jensen. Effective caching of shortest paths for location-based services. SIGMOD, 2012.
[10] A. D. Zhu, H. Ma, X. Xiao, S. Luo, Y. Tang, and S. Zhou. Shortest path and distance queries on road networks: towards bridging theory and practice. SIGMOD, 2013.