Open Geospatial Data, Software and Standards

SOFTWARE Open Access

OSHDB: a framework for spatio-temporal analysis of OpenStreetMap history data

Martin Raifer*, Rafael Troilo, Fabian Kowatsch, Michael Auer, Lukas Loos, Sabrina Marx, Katharina Przybill, Sascha Fendrich, Franz-Benjamin Mocnik and Alexander Zipf

Abstract

OpenStreetMap (OSM) is a collaborative project collecting geographical data of the entire world. The level of detail of OSM data and its data quality vary much across different regions and domains. In order to analyse such variations it is often necessary to research the history and evolution of the OSM data.
The OpenStreetMap History Database (OSHDB) is a new data analysis tool for spatio-temporal geographical vector data. It is specifically optimized for working with OSM history data on a global scale and allows one to investigate the data evolution and user contributions in a flexible way. Benefits of the OSHDB are for example: to facilitate accessing OSM history data as a research subject and to assess the quality of OSM data by using intrinsic measures.
This article describes the requirements of such a system and the resulting technical implementation of the OSHDB: the OSHDB data model and its application programming interface.

Keywords: OpenStreetMap (OSM), OpenStreetMap history data, Data analysis, Data quality assessment, Volunteered Geographic Information (VGI)

Introduction

OpenStreetMap (OSM) is a global, free and openly accessible database of geographical features [1]. It is a popular example of projects that create Volunteered Geographic Information (VGI) [2, 3], where anyone can contribute to the data and the underlying data scheme through collaborative participation [4–6]. The size of the OSM project in terms of the amount of data, the number of contributors, users and applications building on the data has grown considerably since the beginning of the project in 2004. The dataset has undergone constant changes and its geometrical and attributional details are continually evolving. This process of data refinement is spatially heterogeneous, resulting in regions of varying data quality.

OSM data is used in many public, private and commercial areas, e.g., traffic management and logistics, civil protection and disaster management, environmental research and education. In order to evaluate whether the data satisfies the requirements of such applications, researchers have developed methods for data quality assessment. These methods support decision makers in the public, economical and political sectors. Besides data quality, other research fields related to OSM and VGI data in general are addressed as well: How can one gain a better understanding of social and geographical processes that have left patterns in the VGI data? How does a community of such a VGI project work? Who contributes what kind of data when and why?

The aforementioned research questions share a common characteristic: they can be approached by examining the evolution of the data. One advantage of the OSM project is that it provides almost the entire editing history of its global data evolution. This dataset is large and, thus, generally hard to handle. Consequently, it is not utilizable for many potential users. Although many software tools [7–25] address this problem partially, none of them is able to provide the following range of functionality in one single integrated tool: fast and easy reconstruction of historic geometric and semantic states of the data as well as fast spatial, temporal and tag queries. Therefore, we developed

*Correspondence: martin.raifer@uni-heidelberg.de
GIScience Research Group, Institute of Geography, Heidelberg University, Im Neuenheimer Feld 348, 69120 Heidelberg, Germany

a framework called *OpenStreetMap History Database* (OSHDB). It is capable of handling this large dataset and provides the necessary tools to perform Online Analytical Processing (OLAP) tasks related to the history of OSM data in a user-friendly and scalable way.

The OSHDB provides fast data access as well as flexible analysis methods. This is achieved by transforming the original OSM history file into a custom data structure that is tailored towards efficient storage and retrieval of any spatio-temporal extent of the data. This data structure is independent of the storage backend, such that the OSHDB can be adapted to any 3rd-party key-value store. Therefore, the framework can be deployed on a single computer, as well as in a distributed computing cluster. Thus, it can easily be deployed for different usage scenarios.

The remainder of this article is structured as follows: The "Related work" section discusses, in particular, literature about intrinsic data quality, user behavior and related topics, as well as software tools that provide access to the full history data of OSM. The "Implementation" section describes the design goals, components, data model of the OSHDB and how to get access to the data via the OSHDB-API. The "Discussion" section reviews the presented framework and its related technologies. "Conclusions" section presents a summary and outlook towards future research and developments.

Related work

In recent years, several studies utilized the history of the OSM dataset, e.g., for conducting intrinsic quality assessments in cases where comparison datasets do not exist [26, 27]. Intrinsic quality indicators have been developed to evaluate the fitness for purpose of OSM data [13, 17, 18, 24]. Further studies assess the behavior of OSM contributors and contribution patterns [5, 28–31].

Reviewing the literature reveals a need for tools that allow flexible access to the OSM history on a global scale. In the following, an overview of existing software tools and web applications tackling this challenge is provided. We thereby distinguish between tools that allow user-defined queries on the historical data and platforms that allow to generate predefined statistics.

OpenStreetMap Stats [7], OSM stats [8] and OSM Tag History [9] are web applications that present aggregated statistics on a global and/or country level. Other web applications focus on visualizing the influence of contributors. Crowd Lense [10] enables its user to explore OSM changesets and additional contributor information like the preferred language for selected cities. Further, OSMvis [32] facilitates making sense of the evolution of geometries and the folksonomy, as well as analysing OSM changesets and contribution statistics. Another web tool, which can be used to analyse user behavior is "Who did it?" [11]. It creates predefined statistics about changesets, or users on a city level.

The web interface of the tool osm-deep-history [12] provides a view on all versions of any selected node or way. Minghini et al. [13] developed a web interface for visual exploration of the up-to-dateness of OSM nodes or ways in a small area. It makes use of the timestamps in the data, which are referring to the latest edit of the respective element, but does not incorporate the full-history information. For exploring spatial characteristics of the historical OSM data, the web application OSMatrix [14, 15] visualizes statistics like the evolution of numbers of buildings or lengths of roads over time, aggregated on hexagonal grids. OSM analytics [16] allows the user to define a bounding box and shows a graph of the mapping activity in the defined region. Furthermore, the historical state of OSM data at different points in time can be compared within one map view. The presented web applications are designed to visualize characteristics of the OSM history based on predefined parameters, but do not allow comprehensive evaluations regarding data quality. Further limitations are that the OSMatrix is not available globally and OSM analytics does not support complex OSM data types such as multipolygon relations.

Besides web applications, some software tools offer a predefined set of quality indicators. Barron et al.'s iOSMAnalyzer [17] utilizes OSM's full history data to create statistics, charts and maps that evaluate the data quality concerning different categories, such as "general information on the study area" or "routing and navigation". Any extract of the full history dump can be imported and utilized within this tool using the import functionalities of the osm-history-renderer [33]. However, the analyses are limited to smaller regions due to performance limitations. Another set of intrinsic quality indicators is made available by Sehra et al. [18] who implemented an extension of a QGIS processing toolbox to assess the completeness and road navigability of OSM data. Nevertheless, this study is limited to analysing line type features and compares only two points in time. [18]

Moving on to flexible analysis-tools, Osmium [19] is a framework that enables the user to work with OSM data, e.g., by providing conversion functions between several file formats. It also allows the extraction of parts of the data through specified attribute filters or spatial extents. Besides working with regular OSM data, Osmium also offers support for history files and contains functions to extract historical data at any specific point in time. Another tool for querying historical OSM data is the Overpass API [20]. It is designed for returning small subsets of the OSM data in a short amount of time, but only partially supports analysing the full history of an OSM object. It is, for example, able to export single snapshots

of the history. Osm2orc [21] is a tool, which transforms data from the original osm-pbf format into the Apache ORC data structure in order to import it into the Amazon Athena database system. Although being very flexible in querying historical and non-historical data, getting the count of buildings in a specific region with a certain tag requires the user to perform an SQL join operation between OSM node and way entities [34].

In contrast to the presented multi-purpose frameworks using normal, as well as historical OSM files, osm-data-classification [22] was designed to analyse the data through time in order to classify contributors into categories such as beginners, intermediate users, or experts. Data quality is assessed by assuming that the quality of an OSM object is good, if its last contributor is experienced. Limitations of this project are, e.g., long processing times and high memory requirements. With the software framework Epic-OSM [23] by Anderson et al. edits within OSM can be analysed to understand who was mapping where and at what time.

Rehrl and Gröchenig analysed user behavior by looking at different forms of activity [24]. Their framework applies a theoretical model of activities onto OSM data, e.g., for identifying different types of activity levels. OSMesa [25] is another multi-purpose framework for processing OSM data. It is, for instance, used by the Missing Maps leaderboard [35] to generate user statistics. OSMesa utilizes the Apache Spark big data processing framework to compute, e.g., the number of added buildings or roads, or to create vector tiles.

The presented studies and frameworks show that analyzing the OSM history is, in general, of high interest to different fields of academia, community and humanitarian organizations alike. The heterogeneity and the large amount of the data form obstacles when making use of the available tools. In fact, the tools suffer from the flexibility and performance necessary for exploring and investigating the data in detail. The OSHDB fills this gap by making it possible to efficiently conduct flexible analyses on OSM's full history dataset.

Implementation

The OSHDB framework has been designed to be appropriate for a large spectrum of potential use cases. This section summarizes the design goals of the OSHDB and explains their impact on the architecture, the design, the data model and the application programming interface (API) of the framework.

Design goals

Our main goal is to provide easy access to the history of OSM data in order to conduct a wide range of spatio-temporal analyses. In particular, we aim at performing dynamic aggregations by varying thematic, spatial and temporal characteristics of the data. As an example of such an aggregation, it shall be possible to query different spatial and temporal scales in order to compare regions in various temporal granularity. These use cases combined with the characteristics of the dataset motivate the following design goals:

Performance: The dataset is large and, thus, requires efficiency in terms of storage size, data access and processing performance. These benefit from a custom-built, compact data representation and from parallelizing computations.

Lossless information: Analysing all aspects of data evolution and editing activities requires the data schema to provide lossless information on historical OSM data by maintaining all properties of the original data including its errors.

Simple, generic API: A general purpose programming language shall be available to the user in order to permit arbitrary analyses. An API that encapsulates the internal data representation supports usability and maintainability.

Local and distributed deployment: The requirements of performing a small regional analysis of community dynamics can be substantially different from those of a global study on data quality. Hence, our framework shall run equally on a single computer as well as on a large cluster. This goal is related to the above-mentioned parallelization.

The above design goals and the resulting requirements have implications on the design and the implementation of our framework. In the following sections, we discuss these implications in more detail.

Overview

In order to achieve the design goals presented in the "Design goals" section, we decided on a layered architecture. Figure 1 shows the role of the OSHDB as a mediating framework between the raw OSM data and arbitrary analysis applications.

The bottom layer is concerned with the data storage. By relying on third party key-value stores, we are able to deploy the OSHDB both locally and in a distributed cluster environment. Currently, two different backends have been implemented: A generic JDBC (Java Database Connectivity) backend, which can be used with any JDBC-compliant database management system, and a backend for the distributed database and processing platform Apache Ignite [36], which supports massive parallelization. However, by implementing further backends, the OSHDB can be adapted to any key-value store.

The middle layer handles the representation of the OSHDB data. We designed a custom data schema that represents OSM history data compactly and permits fast

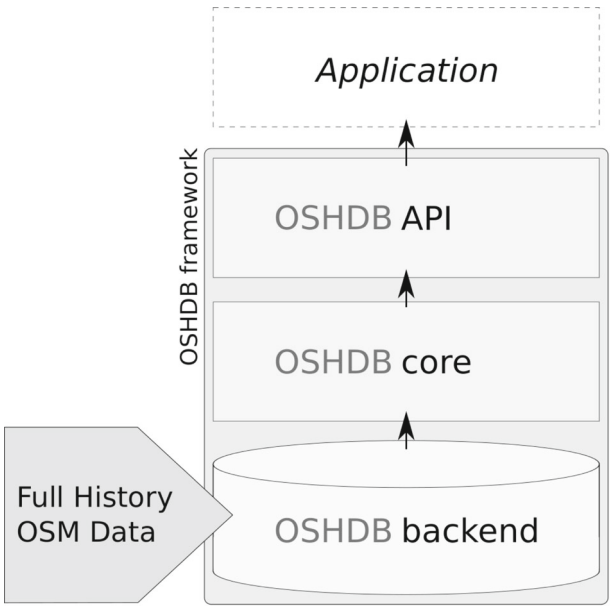

Fig. 1 Overview of the data flow in the OSHDB framework

data access and parallel processing of the data at the same time (see the "Data model" section). This core layer provides means to access the information contained in the binary data returned by the backend databases.

The top layer is formed by a simple, generic API (see the "Application programming interface" section). This API exposes the MapReduce programming model into Java as a general purpose programming language and abstracts from internal data representations and processing strategies. It offers flexible analysis functionalities and allows one to query arbitrary spatio-temporal statistics about the OSM full history data.

Data model

The OSHDB data model is designed for efficient storage of and access to OSM history data. It is compact in size to optimally utilize the available memory. In order to ensure the scalability of the system, the data model also includes a partitioning schema which allows distributed data storage and the parallel execution of computations.

The OSM data model

The data model defined by the OSM project primarily consists of three types of elements: nodes, ways and relations [37]. A *node* defines a point in space. Nodes are the only elements that carry coordinates. A *way* describes a linear or polygonal geometry by a list of references to nodes. Ways are used to model geographical features such as streets and buildings. A *relation* is an ordered collection of elements that groups nodes, ways and relations to a larger unit. Relations can represent polygons or more abstract information like turn restrictions.

Each element consists of a common set of attributes such as an ID, a modification timestamp, a version number, the ID of the modifying user and a list of attribute tags in form of key-value pairs. In addition, ways and relations contain the IDs of their members as references. In order to construct the geometry of a way or a relation these references have to be resolved because only nodes actually contain coordinates.

OSH entities

When taking the modification history into account, several OSM elements may have the same ID. These elements can be distinguished by their version number, which is incremented at each modification. We group all elements belonging to the same ID into a so-called OSH entity. The elements of such an entity are identified through their version number. For each OSM element type (node, way and relation) a corresponding OSH type exists.

As shown in Table 1, the current OSM history database consists of approximately 8.4 billion versions for 6.1 billion entities. The average number of versions increases from nodes over ways to relations. The modifications of the members of a way or relation are not directly reflected in the reference-lists containing these members, i.e., in order to resolve a reference, all versions of the referenced elements need to be considered. Thus, ways and relations may have many more implicit versions than their version number indicates. Therefore it is reasonable to store all versions of an entity together in one location.

This grouping of versions with the same ID renders it possible to efficiently apply a delta encoding between consecutive versions. An example is shown in Table 2. Since small numbers can be stored with fewer bits than large ones, it is beneficial to store the difference between the timestamps of two consecutive versions instead of the actual timestamps. Similarly, we only store which references and tags change between versions instead of repeating the complete information in each version. This delta encoding leads to a compact representation of the data, in particular for potentially large entities like ways and relations, which tend to have a high number of versions per ID. As a downside of this encoding, it is impossible to extract a specific version from an OSH entity without

Table 1 Number of versions and unique IDs in a typical OpenStreetMap full history planet file

OSM element	Versions	Unique IDs	∅
Nodes	7 326 018 355	5 540 766 395	1.32
Ways	1 002 978 129	596 015 042	1.68
Relations	25 767 100	7 796 460	3.30

Column ∅ contains the average number of versions per unique ID for each of the OSM element types. These numbers have been calculated for the OSM full history planet file `history-181112.osm.pbf`, downloaded from https://planet.openstreetmap.org, accessed 2018-11-19

reading all previous versions. By sorting the versions from the latest to the earliest, we optimize for queries that only need the latest version or multiple versions at once. In contrast, if each OSH entity was stored as an array of OSM elements, one could directly access each version of the entity for the cost of increased storage size.

Partitioning

We partition the dataset into independent subsets and store them in a key-value database as an underlying data storage system. This makes it possible to distribute the data and to process data partitions in parallel.

For efficiency, each subset is completed by referenced entities. For example, in order to analyse an OSH way that references many OSH nodes, the way itself as well as all its referenced nodes need to be considered and, thus, are stored in the same data partition. This principle is also applied to OSH relations referencing multiple OSH nodes, ways, and relations. As a result, parts of the data are stored in several partitions, which results in duplicates. We keep track of these duplicates to avoid negative side effects such as counting entities multiple times.

The OSHDB does not enforce a particular partitioning schema, but as a majority of queries are expected to use a spatial extent, we provide a spatial partitioning schema: Spatial extents of OSM data elements vary a lot from small features like single trees or postboxes, to geometries of large extent, such as country borders or international road routes. To handle both of these extremes the OSHDB implements a partition schema that makes use of a spatial grid with a number of zoom levels. Each OSH entity is, according to the schema, stored in that grid cell of a specific zoom level that fits best to the bounding box of the entity. This typically results in the storage of small entities in higher zoom levels, while large entities are stored in lower zoom levels.

In order to further reduce computation times of queries with a spatial filter, each of the resulting partitions additionally includes a local spatial index of the contained OSH entities. Figure 2 summarizes the different parts of the data model used by the OSHDB.

Table 2 Example of the delta encoding in OSH entities

OSM

```
osm-way {
  id: 1234,
  version: 3,
  uid: 10,
  timestamp: 1000,
  tags: [k1:v1,k2:v2;k3:v3],
  refs: [1,2,3,4]
}
osm-way {
  id: 1234,
  version: 2,
  uid: 10,
  timestamp: 900,
  tags: [k1:v1],
  refs: [1,2,3,4]
}
osm-way {
  id: 1234,
  version: 1,
  uid: 5,
  timestamp: 400,
  tags: [k1:v1],
  refs: [1,2,3]
}
```

OSH

```
osh-way {
  id: 1234,
  versions: [{
    version: 3,
    uid: 10,
    time-delta: 1000,
    tags-added:
    [k1:v1,k2:v2,k3:v3]
    refs: [1,2,3,4]
  }, {
    time-delta: 100,
    tags-removed:
    [k2:v2,k3:v3]
  }, {
    uid: 5,
    time-delta: 500,
    refs-removed: [4]
  }]
}
```

Application programming interface

The application programming interface (API) of the OSHDB framework makes it possible to build custom applications for analysing OSM history data. Its design and implementation are described in this section.

Requirements

In the context of general purpose OSM data retrieval and analysis, different needs arise. Questions are often complex and involve more than a predefined simple query. The optimal way of retrieving the desired data strongly depends on the type of the query: Do we count or sum specific features? Is a logical or statistical argumentation necessary? Which thematic aspects do we focus on? etc. In addition, OSM data is too extensive to be easily processed on a single-core machine. Such needs create, in particular, the following requirements:

Abstraction from the data model: The concepts used internally to store and access the data shall remain hidden in order to liberate the user from having to deal with this additional complexity.

Filtering and aggregation: The current data model of OSM represents the environment by a large number of elements and their respective changes. For further processing, the data shall be made available in an efficient way by providing the functionality to filter and aggregate the data.

Affording parallelization: OSM data is increasingly growing, creating the need to process the data efficiently on multi-core systems.

Distributed and local execution: Queries involve a variety of scales. The API should afford performing global queries on a server cluster, while it should also afford performing local queries on a single machine only.

In the following, we discuss the design of the API used to access the OSHDB.

API design

The needs and requirements of the API can only be met in their generality by the use of a general purpose

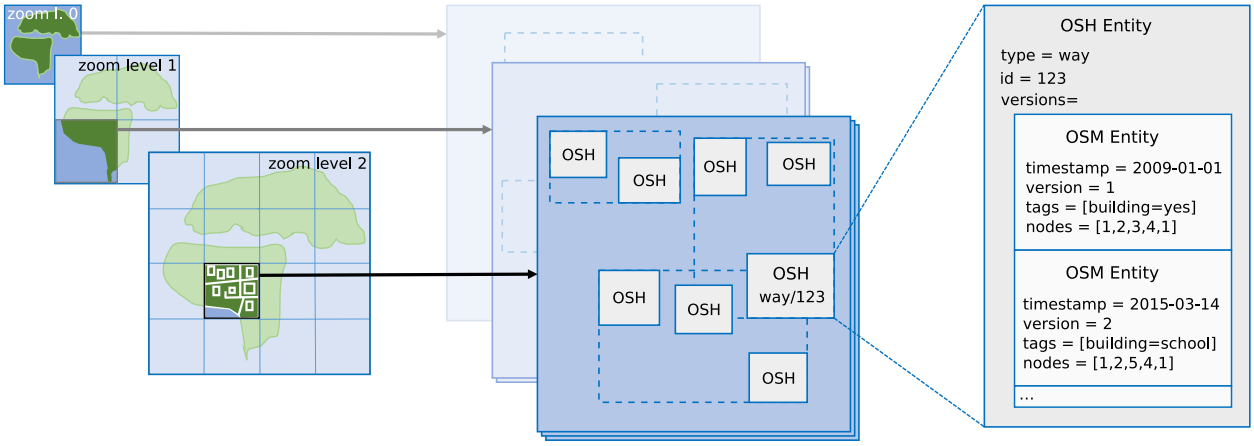

Fig. 2 A schematic overview of the parts of the data model used by the OSHDB. The global OSM full history data is partitioned into cells of varying size. Each of these partitions contains information about OSH objects, which are collections of all versions of an individual OSM element

programming language. A multiplicity of approaches to APIs exist, among them visual interfaces of varying nature, web APIs offered via the HTTP protocol with typically limited functionality, domain specific languages, and methods offered as part of an existing general purpose programming language. The latter provides a high flexibility when treating general questions without restriction to some few domains. Other ways of accessing the data can be implemented on top of such a flexible API because they offer less functionality.

The OSHDB is optimized for efficiently retrieving OSM history data, which requires the use of indices, optimized data types, and internal methods. While such internally used concepts are important for the efficiency and stability of the database, they are, by and large, not of interest for the user. The API is thus solely built on the concepts of the OSM data model rather than exposing the internally used concepts to the user. Such use of only well-known concepts renders it possible to employ the API without the need to learn many new concepts – the learning curve is shallow.

The parallelization of algorithms is hard to achieve in general, in particular when the user is not familiar with corresponding concepts. Side effects can, e.g., only be parallelized with good knowledge about when side effects occur and if the processes running in parallel are synchronized in some way. The functional programming paradigm prohibits side effects when followed strictly. In addition, it can be traced in which order computations need to be executed, and which computations can be performed in parallel – such tracing is often performed using abstract syntax trees in case of functional programming languages. The OSHDB-API follows the functional paradigm, which is why it allows a straight forward implementation of the MapReduce principle [38–40] in the backend.

Monads [41] offer a great possibility to build processing pipelines. Starting with an initial state, which incorporates the entire OSHDB, several commands can be executed. Each such command performs an operation on the internal state, e.g., by filtering the OSM history data, by mapping the internal state, or by summarizing the results. Many such commands can be chained, which allows to efficiently divide the required computation into small pieces. At the end of such a chain, the internal state, i.e., the processed data, is returned. Such statements returning the internal state can, e.g., provide a list or a sum. Further, the operations discussed here – the views for starting the processing pipeline, the commands for transforming the data, and the methods that return the result – comply to the formal concept of a monad. The API is based on the concept of a monad, because it allows for a flexible handling of the data in the backend.

Implementation of the API

Starting from the low level data structures described in the "Data model" section, the API – called *OSHDB-API* – builds a few layers of abstraction. First the actual data storage system is specified in a database backend, then (depending on the respective underlying database system) one of several different algorithms for workload distribution are executed. For example, on a local system with random access to the OSHDB data cells, a different algorithm is applied than on a cluster environment, in which the data is distributed across different machines. On top of this, the API provides different *views* on the data. These

views provide a more accessible and concrete interpretation of the base data: Instead of the raw OSM data entities, the user is able to access the data via refined data structures that contain concrete geometries following the simple features standard [42] and contextual metadata. Finally, a layer of map-reduce methods provides access to these views, which can finally be used to implement custom analysis code.

Database backends An OSHDB database backend specifies where the data is stored and how it can be retrieved. Currently, the OSHDB-API implementation supports two different database backends: a) A generic JDBC backend, which can access OSHDB data stored in any JDBC-compatible database, for example the lightweight local file-based database system H2 and b) a backend that handles access to OSHDB data stored in a distributed computation cluster running the Apache Ignite [36] software. The OSHDB can be adapted to work with any key-value store by implementing additional backends.

Workload distribution Depending on the underlying data storage system, the methods for executing analyses are implemented in different ways. On a local system, already existing process threads can be used to process the data fetched from a local database. In contrast, the code has to be transferred to the remote nodes of the computing cluster, where they can be run collocated with the underlying data. Maximizing the collocation of the data and their computations reduces the amount of data that has to be transferred over network connections, which would otherwise reduce the performance benefits of a parallel processing environment [43].

In general, for a given database backend, such a workload distribution can be implemented in different ways. While one algorithm might be preferred in some use cases (e.g., a query over a relatively small area), there might exist alternative implementations that work better in other scenarios (e.g., a global analysis query). To satisfy several of these use cases, the OSHDB-API allows a database backend to have multiple workload distribution implementations, which can be activated according to the different needs of a user.

Abstraction from data model to API views One of the premises of the OSHDB-API is that an end user typically does not want to work directly on the raw OSM history data, but rather wants to ask questions about the geographical features that are represented by the data. Also some of the filtering and data-aggregation options listed in Table 3 are only meaningful when applied on actual geographical features that have a geometric extent and shape and which exist at certain points in time. For this reason, the API provides refined data objects that contain concrete geometric representations of OSM entities and contextual metadata (e.g., about the point in time for which the respective geometric representation is valid) alongside the underlying OSM history data itself.

Internally, this step consists of two different parts: One part is responsible for generating the spatial representation (following the simple features specification [42]) of a given OSM entity at a given point in time. While, by default, this module tries to closely follow the rules defined in the OSM wiki [1] about how to generate these geometries, it still remains configurable to allow different geometric interpretations of the OSM data for special use cases, e.g., to be able to also work with uncommon or undocumented tags. The other part deals with the interpretation of the temporal aspect of the data: using the OSHDB-API it is possible to look at the historical development of the data in two fundamentally different ways, called *views*:

Snapshots: This view returns how the underlying OSM data looked like at specified point(s) in time.

Contributions: This view returns all modifications of the underlying OSM data in the specified time interval(s).

Depending on the used view, a different stream of data is provided. This stream can be used in consequent map-reduce steps for processing the data and finally computing the desired result. The snapshot view returns at most n snapshots (where n is the number of requested time slices) of each matching OSM entity. Each of these snapshots contains information about the timestamp for which this snapshot has been obtained, the geometry of the underlying OSM entity and the entity's raw data. The contribution view returns a number of contribution objects. This includes direct modifications of the corresponding OSM entities (e.g., when a new tag is added to an existing feature) as well as indirect modifications originating from changes to the entity's referenced members (e.g., when the coordinates of a way's nodes are altered). Thereby, access is provided to the following properties: point in time when the change happened, contribution-type, geometry and state of the respective entity before and after the modification, changeset ID and user ID associated with the change, and the raw data of the underlying OSM entity. Here, the contribution-type can be either a creation of an entity, a deletion of an entity or one of several different types of modifications of an entity.

These views can be used for computing answers to different types of questions about the OSM data. For example, to calculate the historical development of the length of certain types of roads in OSM the snapshot view can

Table 3 Methods offered by the application programming interface of the OSHDB framework

Method	Description
Data Filters	
`areaOf-Interest`	Sets the geographical area of interest of the query, i.e. a bounding box or a bounding (multi)polygon.
`timestamps`	Sets the temporal limits of the query. Depending on the analysis view, this can either represent a list of independent timestamps (*snapshots*) or represent a list of time intervals.
`osmType`	Filters data by their OSM entity type, i.e. a *node*, *way* or *relation*.
`osmTag`	Filters data by their OSM tags. Can either filter for `key=*` presence of a tag with a given key, `key=value` presence of a specific tag (key-value combination), `key=[values]` presence of a tag whose value is in the given list of values, `key regex` presence of a tag whose value matches the given regular expression, `[key=value]` presence of at least one tag from the given collection of key-value pairs.
Aggregation Methods	
`aggregateBy`	Defines a custom aggregation method. When applied on a MapReducer, it transforms it into a new MapAggregator with the same settings as the original MapReducer, but which processes the input data in chuncks defined by the indices returned by the given aggregate-by function. When applied on a MapAggregator, the already existing aggregation indices are refined further by the new aggregate-by function (where the resulting index set is defined as the cross product of the existing and the new set of indices).
`aggregateBy-Timestamp`	Aggregates results by a temporal index. This knows about the overall query timestamps parameter and if necessary associates timestamps to the respective time intervals defined for the whole query.
`aggregateBy-Geometry`	Aggregates results by their geometries (spatial position and extent). Accepts a set of arbitrary (multi)polygons which define the aggregation index. If necessary, this method splits and clips the geometries of OSM entities, when they extend over multiple polygons.
MapReduce Methods	
`map`	Performs a data transformation step that calculates one output object for each input object of the processing stream.
`flatMap`	Performs a data transformation step that calculates arbitrarily many output objects for each input object of the data stream and flattens the resulting output object in the processing stream.
`filter`	Filters the processing stream by the given predicate.
`reduce`	Performs a generic reduce operation of the processing stream, which ultimately generates a single result object for the entire processing stream.
Specialized Reducers	
`sum`	Calculates the sum of all values.
`count`	Returns the number of entries in the processing stream.
`uniq`	Returns the set of unique values in the processing stream.
`countUniq`	Returns the number of unique entries in the processing stream.
`average`	Calculates the average of all values.
`weighted-Average`	Calculates a weighted average over all values.
`estimated-Median`	Returns an estimation of the median of all values in the processing stream, using the T-Digest method [50].
`estimated-Quanitle(s)`	Returns an estimation of the quantile(s) of the distribution of all values in the processing stream, using the T-Digest method [50].
`collect`	Returns a list of all objects in the processing stream. See the `stream` method below, for a less memory intensive variant of this.
Other	
`stream`	Returns all values as a (JAVA) stream. Equivalent to the `collect` reducer, but doesn't need to buffer the whole dataset in memory before returning.
`groupByEntity`	Special `map` function that groups consecutive entries of the processing stream together which belong to the same original OSM entity.

be used. The contribution view can, in contrast, be used to determine the total number of contributors who have been working on mapping these roads.

Aggregator classes On top of this foundation of data storage, workload distribution and data interpretation, there exists a framework layer, which provides the actual

interface to a user writing queries using the OSHDB-API. The overall design of this layer is inspired by the typical MapReduce implementations found in many programming languages, specifically the `Stream` class that was introduced in Java 8. It provides means to transform, filter, partition, and aggregate the stream of data produced by the underlying layers of the API. The methods that are offered by OSHDB's API are listed in Table 3. Generally, these methods can be divided into different categories:

> *Settings and data filters* are used to select a subset of OSM data and the points (or intervals) in time for which the analysis query should be performed.
> *Aggregation methods* can be used to define how the data is partitioned when calculating results. For example, `aggregateByTimestamp` can be used to get individual results for each requested timestamp.
> *MapReduce methods* are used to transform, to filter and, to compute the final result(s).
> *Specialized reducers* exist to make frequently used reduce operations more convenient: they include, among others, shorthand methods to count, sum, or average the stream of data values.
> *Other methods* include advanced query customization options.

After partitioning, the data type of the result of any reduce operation, such as the generic `reduce` or its shorthands like `count` and `sum`, changes from being a single value to an associative array of values. For instance, instead of a simple integer count, one receives an independent count for each requested timestamp after applying `aggregateByTimestamp`. In order to keep the API consistent across non-partitioned and partitioned modes of operation, we decided to collect the non-aggregating and aggregating reduce methods into separate classes called `MapReducer` and `MapAggregator`, respectively.

Examples

In the following, two basic examples of analysis queries implemented in JAVA using the OSHDB-API are presented.

The first query returns the number of distinct OSM contributors who modified OSM entities that are tagged as `highway` (e.g., motorways, roads, tracks, or paths) and lie in the polygon defined in the variable `region`. The variable `oshdb` is a connection object that contains information about where the OSHDB data is stored and which database backend the query should use. This query uses the `OSMContributionView`, which allows for iteration over all modifications of the underlying data. After mapping each contribution object to the respective contributor's user ID, the final reduce step `countUniq` provides the desired result.

```
Integer numberOfUsersEditingHighways =
OSMContributionView.on(oshdb)
  .areaOfInterest(region)
  .timestamps("2007-10-07", "2018-11-14")
  .osmTag("highway")
  .map(getContributorUserId)
  .countUniq();
```

The second example shows how to access data about the evolution of the length of the network of `highway` objects in OSM over time. Here, a monthly time interval is defined in `timestamps` and the `aggregateByTimestamp` functionality is used. Then, the result of this query contains for each requested timestamp the sums of highway lenghts at each particular point in time.

```
SortedMap<OSHDBTimestamp, Number>
highwayLengthOverTime =
OSMEntitySnapshotView.on(oshdb)
  .areaOfInterest(region)
  .timestamps("2007-10-07", "2018-11-14",
    Interval.MONTHLY)
  .osmTag("highway")
  .aggregateByTimestamp()
  .map(getLengthInMeters)
  .sum();
```

Requirements revisited

The requirements outlined in the "Requirements" section and used for motivating the OSHDB-API in the "API design" section are met. The *abstraction from the data model* is fulfilled by the base abstractions that handle the raw database access. Additionally, the OSHDB-API provides an abstraction layer that refines the raw historical OSM data into views, which are useful for concrete analysis applications. *Parallelization* as well as *distributed and local execution* are governed by the different workload distribution algorithms implemented in the API. These cover the entire spectrum from the execution of a query using a local OSHDB database, to the massively parallelized and distributed execution on a cluster of computers. The resulting user-facing layer of the API itself is designed to afford intuitive and flexible *filtering and aggregation* of the data and is based on principles from the functional programming paradigm such as the MapReduce model of transforming, partitioning, and reducing data.

Discussion

One could argue that the OSHDB is a member of a new class of spatio-temporal data analysis frameworks: it

spans multiple scales, is able to process the full variety of feature types in OSM and addresses OSM's heterogeneity in spatial and temporal resolution. Analysing historical OSM data in its entirety requires such a system. Multiple approaches have been conducted by several groups. These are, however, mostly found to be restricted in at least one of the following points: a) the maximum amount of data that can reasonably be processed and b) the range of different analysis questions that can be answered.

Since the scope of the OSHDB is not to directly analyse raw OSM data elements, other OSM data analysis frameworks potentially perform better in certain situations. This is, for example, the case when querying statistics about the raw OSM data itself. However, the OSHDB covers a broader range of applications, which benefit from the refined data structures that are made available through its API.

Similarly for some other use cases, specifically designed solutions exist that are better suited for their respective application area than the OSHDB, but do not generalize well to other analyses. On the other side of the spectrum are systems that try to be more generic but sometimes limit performance or have restrictions in the (spatial or temporal) scale of queries that can be executed.

The OSHDB framework focuses on combining high performance with flexible data querying capability and generic data analysing functionality. Its compact data format allows fast access to the information provided by the OSM full history data. Its API allows to program analyses in an intuitive and generic way. These queries can be executed both on local systems or on large distributed compute clusters.

The versatility of the OSHDB is shown by the fact that it is already used in a variety of applications: in ongoing research projects about intrinsic data quality assessment [44, 45], for scientific studies related to disaster management [46] and data quality [47], or to run a public web API [48, 49] that provides statistics about the evolution of the OSM data.

Conclusions

OSM data plays an important role as a provider of free and open geographical data about the world. It is of continuing interest to analyse the evolution and the quality of the data. The "Related work" section reviews existing software for analysing OSM data in which we identified a gap in available software tools in regards to flexibility and performance/scalability, which is filled by the here presented OSM data analysis framework OSHDB.

Its design goals, and an overview of the technical implementation is given in the "Implementation" section: The OSHDB follows a modular approach: on top of a specially designed data model, a map-reduce based API allows users to implement their own data analysis applications.

The OSHDB also allows parallelized computation on a distributed compute cluster. As has been discussed in the "Application programming interface" section, the API is flexible and thus not tailored to a specific application. The API can rather be used for a broad range of applications. Examples for such applications are intrinsic data quality assessment, or creating large scale data visualizations.

There exist still many areas in which the OSHDB can be improved in the future. For example, performance can be further increased through refinements of the data model, e.g., by implementing more optimized data partitioning schemes for different use cases. Our future plans include improving the API by broadening its application domains, for example through adding the possibility to perform spatial joins in analysis queries.

Availability and requirements

Project name: OSHDB
Project home page: https://github.com/giscience/oshdb
Operating systems: Platform independent
Programming language: Java
Other requirements: Java 8 or higher
License: GNU Lesser General Public License

Abbreviations

API: Application programming interface; JDBC: Java database connectivity; OLAP: Online analytical processing; OSM: OpenStreetMap; VGI: Volunteered geographic information

Acknowledgements

The authors would like to thank the OpenStreetMap community for providing the open and free OSM data.
Further, we would like to thank the GIScience Research Group, Heidelberg University, for providing helpful comments on the OSHDB development, beta-testing of the software and their contributions to the OSHDB source code.

Funding

The work presented in this paper has been funded by the Klaus Tschira Foundation. Additionally, Franz-Benjamin Mocnik has been funded by Deutsche Forschungsgemeinschaft as part of the project *A framework for measuring the fitness for purpose of OpenStreetMap data based on intrinsic quality indicators* (FA 1189/3-1).
We acknowledge financial support by Deutsche Forschungsgemeinschaft within the funding program Open Access Publishing, by the Baden-Württemberg Ministry of Science, Research and the Arts and by Ruprecht-Karls-Universität Heidelberg.

Availability of data and materials

OSHDB's source code is accessible on GitHub: https://github.com/giscience/oshdb. Compiled versions of OSHDB releases are available as maven packages on https://repo.heigit.org. OSHDB data is available for download on https://downloads.ohsome.org. Further documentation is available on https://docs.ohsome.org.

Authors' contributions

The authors worked collectively on the development of the OSHDB software and this article. AZ initiated the project and contributed to the design goals of the OSHDB. RT, LL and MA developed the OSHDB data model. MR and F-BM designed and implemented the application programming interface. KP, SM and FK performed the review of related work. The "Introduction" section was written by MA, LL and SF. The "Related work" section is based on a literature and software review by KP and was written by FK and SM. The "Design goals" section was written by SF, LL and MA. The "Overview" section was written by SM and

MR. The "Data model" section was written by RT and SF. The "Application programming interface" section was written by F-BM ("Requirements" and "API design" sections) and MR ("Implementation of the API" and "Requirements revisited" sections). The "Discussion" section was written by MR. The "Conclusions" section was written by MR. FBM and SF provided helpful comments regarding all sections of this article. All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note
Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Received: 26 November 2018 Accepted: 6 March 2019
Published online: 08 April 2019

References

1. OpenStreetMap Wiki. https://wiki.openstreetmap.org. Accessed 13 Nov 2018.
2. Haklay M. How good is volunteered geographical information? A comparative study of OpenStreetMap and Ordnance Survey datasets. Environ Plann B. 2010;37(4):682–703. https://doi.org/10.1068/b35097.
3. Goodchild MF. Citizens as sensors: the world of volunteered geography. GeoJournal. 2007;69:211–21. https://doi.org/10.1007/s10708-007-9111-y.
4. Mocnik F-B, Zipf A, Raifer M. The OpenStreetMap folksonomy and its evolution. Geo-Spatial Inf Sci. 2017;20(3):219–30. https://doi.org/10.1080/10095020.2017.1368193.
5. Arsanjani JJ, Barron C, Bakillah M, Helbich M. Assessing the quality of OpenStreetMap contributors together with their contributions. In: Proceedings of the AGILE. Leuven; 2013.
6. Mooney P, Corcoran P. The annotation process in OpenStreetMap. Trans GIS. 2012;16(4):561–79. https://doi.org/10.1111/j.1467-9671.2012.01306.x.
7. OpenStreetMap Stats. https://osmstats.stevecoast.com/dashboard/ht/total. Accessed 13 Nov 2018.
8. Neis P. OSMstats. https://osmstats.neis-one.org. Accessed 13 Nov 2018.
9. OSM tag history. http://taghistory.raifer.tech/. Accessed 13 Nov 2018.
10. Crowd Lense. http://sterlingquinn.net/apps/crowdlens. Accessed 13 Nov 2018.
11. Zverev, I. Who did it? http://zverik.openstreetmap.ru/whodidit/. Accessed 13 Nov 2018.
12. osm-deep-history. www.github.com/osmlab/osm-deep-history. Accessed 13 Nov 2018.
13. Minghini M, Brovelli MA, Frassinelli F. An open source approach for the intrinsic assessment of the temporal accuracy, up-to-dateness and lineage of OpenStreetMap. ISPRS - Int Arch Photogramm Remote Sens Spat Inf Sci. 2018;XLII-4/W8:147–54. https://doi.org/10.5194/isprs-archives-XLII-4-W8-147-2018.
14. Roick O, Hagenauer J, Zipf A. OSMatrix — grid-based analysis and visualization of OpenStreetMap. In: State of the Map Europe 2011. Vienna; 2011. p. 44–54.
15. Roick O, Loos L, Zipf A. A Technical Framework for Visualizing Spatio-temporal Quality Metrics of Volunteered Geographic Information. In: Geoinformatik 2012: Mobilität und Umwelt. Braunschweig; 2012. p. 263–270.
16. OSM analytics. https://osm-analytics.org/. Accessed 13 Nov 2018.
17. Barron C, Neis P, Zipf A. A comprehensive framework for intrinsic OpenStreetMap quality analysis. Trans GIS. 2014;18(6):877–95. https://doi.org/10.1111/tgis.12073.
18. Sehra SS, Singh J, Rai HS. Assessing OpenStreetMap data using intrinsic quality indicators: An extension to the QGIS processing toolbox. Future Internet. 2017;9(2). https://doi.org/10.3390/fi9020015.
19. osmium. https://osmcode.org/docs.html. Accessed 13 Nov 2018.
20. Overpass API. https://github.com/drolbr/Overpass-API. Accessed 13 Nov 2018.
21. osm2orc. https://github.com/mojodna/osm2orc. Accessed 13 Nov 2018.
22. Oslandia. https://oslandia.com/en/?s=osm+data+classification. Accessed 13 Nov 2018.
23. Anderson J, Soden R, Anderson KM, Kogan M, Palen L. EPIC-OSM: A software framework for OpenStreetMap data analytics. In: 49th Hawaii International Conference on System Sciences (HICSS). Koloa: IEEE; 2016. p. 5468–77. https://doi.org/10.1109/HICSS.2016.675.
24. Rehrl K, Gröchenig S. A framework for data-centric analysis of mapping activity in the context of volunteered geographic information. ISPRS Int J Geo-Inf. 2016;5(3):37. https://doi.org/10.3390/ijgi5030037.
25. OSMesa. https://github.com/azavea/osmesa. Accessed 13 Nov 2018.
26. Barrington-Leigh C, Millard-Ball A. The world's user-generated road map is more than 80% complete. PLoS ONE. 2017;12(8):0180698. https://doi.org/10.1371/journal.pone.0180698.
27. Nasiri A., Ali Abbaspour R, Chehreghan A, Jokar Arsanjani J. Improving the quality of citizen contributed geodata through their historical contributions: The case of the road network in OpenStreetMap. ISPRS Int J Geo-Inf. 2018;7(7):253. https://doi.org/10.3390/ijgi7070253.
28. Neis P, Zipf A. Analyzing the contributor activity of a volunteered geographic information project ? the case of OpenStreetMap. ISPRS Int J Geo-Inf. 2012;1(2):146–65. https://doi.org/10.3390/ijgi1020146.
29. Mooney P, Corcoran P. Analysis of interaction and co-editing patterns amongst openstreetmap contributors. Trans GIS. 2014;18(5):633–59. https://doi.org/10.1111/tgis.12051.
30. Yang A, Fan H, Jing N, Sun Y, Zipf A. Temporal analysis on contribution inequality in openstreetmap: A comparative study for four countries. ISPRS Int J Geo-Inf. 2016;5(1):5. https://doi.org/10.3390/ijgi5010005.
31. Gröchenig S, Brunauer R, Rehrl K. Digging into the history of vgi data-sets: Results from a worldwide study on openstreetmap mapping activity. J Locat Based Serv. 2014;8(3):198–210. https://doi.org/10.1080/17489725.2014.978403.
32. Mocnik F-B, Mobasheri A, Zipf A. Open source data mining infrastructure for exploring and analysing OpenStreetMap. Open Geospatial Data Softw Stand. 2018;3(7). https://doi.org/10.1186/s40965-018-0047-6.
33. osm-history-renderer. https://github.com/MaZderMind/osm-history-renderer. Accessed 8 Jan 2019.
34. osm2orc-blog. https://aws.amazon.com/blogs/big-data/querying-openstreetmap-with-amazon-athena/. Accessed 13 Nov 2018.
35. Missing Maps Leaderboards. http://www.missingmaps.org/leaderboards/. Accessed 13 Nov 2018.
36. Apache Software Foundation. Apache Ignite, a distributed database, caching, and processing platform. 2018. https://ignite.apache.org/. Accessed 13 Nov 2018.
37. OpenStreetMap Wiki, Elements page. https://wiki.openstreetmap.org/wiki/Elements. Accessed 13 Nov 2018.
38. Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. In: Proc 6th Symp Oper Syst Des Implemen (OSDI). San Francisco; 2004. p. 137–50.
39. Chen R, Chen H, Zang B. Tiled-MapReduce: Optimizing resource usages of data-parallel applications on multicore with tiling. In: Proc 19th Int Conf Parallel Architectures Compilation Tech (PACT). Vienna: ACM; 2010. p. 523–34.
40. Cary A, Sun Z, Hristidis V, Rishe N. Experiences on processing spatial data with MapReduce. In: Winslett M, editor. Scientific and Statistical Database Management (SSDBM). Lecture Notes in Computer Science, vol 5566. Berlin: Springer; 2009. p. 302–19. https://doi.org/10.1007/978-3-642-02279-1_24.
41. Wadler P. The essence of functional programming. In: Proc 19th ACM SIGPLAN-SIGACT Symp Princ Program Lang. Albuquerque: ACM; 1992. p. 1–14. https://doi.org/10.1145/143165.143169.
42. Open Geospatial Consortium, et al. OpenGIS implementation standard for geographic information - simple feature access - part 1: Common architecture. OpenGIS implementation standard. Document OGC. 2010;13–40.
43. Amdahl GM. Validity of the single processor approach to achieving large-scale computing capabilities. In: AFIPS Conference Proceedings. Atlanic City: ACM; 1967. p. 483–5. https://doi.org/10.1145/1465482.1465560.
44. Mocnik F-B. Linked open data vocabularies for semantically annotated repositories of data quality measures. In: Proc 10th Int Conf Geogr Inf Sci (GIScience). Melbourne: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik; 2018. p. 50:1–7. https://doi.org/10.4230/LIPIcs.GISCIENCE.2018.50.
45. Mocnik F-B. OSM Measure Repository. 2018. https://osm-measure.geog.uni-heidelberg.de/. Accessed 14 Nov 2018.

46. Auer M, Eckle M, Fendrich S, Griesbaum L, Kowatsch F, Marx S, Raifer M, Schott M, Troilo R, Zipf A. Towards using the potential of OpenStreetMap history for disaster activation monitoring. In: Boersma K, Tomaszewski B, editors. Proceedings of ISCRAM 2018 - 15th International Conference on Information Systems for Crisis Response and Management. Rochester; 2018. p. 317–25.
47. Mocnik F-B, Raifer M. The effect of tectonic plate motion on OpenStreetMap data. In: Proc 21st AGILE Conf Geogr Inf Sci. Lund; 2018.
48. Auer M, Eckle M, Fendrich S, Kowatsch F, Loos L, Marx S, Raifer M, Schott M, Troilo R, Zipf A. Ohsome - eine Plattform zur Analyse raumzeitlicher Entwicklungen von OpenStreetMap-Daten für intrinsische Qualitätsbewertungen. AGIT J. 2018;4:162–7. https://doi.org/10.14627/537647020.
49. ohsome API. https://api.ohsome.org/. Accessed 13 Nov 2018.
50. Dunning T, Ertl O. Computing Extremely Accurate Quantiles Using t-Digests. 2017. https://github.com/tdunning/t-digest/. Accessed 13 Nov 2018.