NoDA: Unified NoSQL Data Access Operators for Mobility Data

Nikolaos Koutroumanis, Panagiotis Nikitopoulos, Akrivi Vlachou, Christos Doulkeridis
Department of Digital Systems
University of Piraeus
Piraeus, Greece
koutroumanis@unipi.gr,nikp@unipi.gr,avlachou@unipi.gr,cdoulk@unipi.gr

ABSTRACT

In this paper, we propose NoDA, an abstraction layer consisting of spatio-temporal data access operators, which is used to access NoSQL storage engines in a unified way. NoDA alleviates the burden from big data developers of learning the query language of each NoSQL store, and offers a unified view of the underlying NoSQL store. Our approach is inspired by the equivalent paradigm of drivers (such as JDBC) in the relational database world, where the application code is indifferent to the exact underlying database engine. Still, the challenges in the NoSQL world are manifold, because of the lack of standardization in data access. We focus on the specific case of mobility data, and show how spatial and spatio-temporal operators, such as range queries and *k*-nearest neighbor, are supported in a unified way. Moreover, we present challenges and solutions for supporting spatial and spatio-temporal data in NoSQL stores.

CCS CONCEPTS

• **Information systems** → **Key-value stores**.

KEYWORDS

NoSQL, data access operators, spatio-temporal data

ACM Reference Format:
Nikolaos Koutroumanis, Panagiotis Nikitopoulos, Akrivi Vlachou, Christos Doulkeridis. 2019. NoDA: Unified NoSQL Data Access Operators for Mobility Data. In *16th International Symposium on Spatial and Temporal Databases (SSTD '19), August 19–21, 2019, Vienna, Austria.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3340964.3340981

1 INTRODUCTION

In the current era, position tracking devices can be found on most moving objects (vessels, vehicles, aircraft, etc.) around the globe. Data management systems handling the tracing information produced by such devices, face a great challenge, since the volume and velocity of mobility data are ever-increasing. The combination of these challenges has led to the need for a different breed of data management systems, supporting different data models as well as scalable distributed storage and querying. These systems, known as NoSQL stores [1], provide increased scalability and availability characteristics.

NoSQL stores have appeared and are being successfully deployed in several applications for almost a decade, thus their maturity has increased over the last years. Many NoSQL stores can now inherently handle geospatial data, by providing dedicated indexes and query types. However, NoSQL stores still rely on heterogeneous languages and there exists no user-friendly, unified query language to date, which can work seamlessly on all NoSQL stores. This shortcoming, also highlighted in [3], may incommode the transition to a NoSQL store, since the application developers need to learn its specific query language. Listings 1.1 and 1.2 show how a simple selection operation ("find all records having values of a field ≥ 5") is performed in Java over MongoDB and HBase, respectively. Clearly, the query language is far from standardized, which puts a burden on developers that must become acquainted with different code syntax.

Listing 1.1: Typical code for a MongoDB filter query.

```java
MongoClient mongoClient = new MongoClient();
MongoCollection m = mongoClient
    .getDatabase("test").getCollection("collection");
FindIterable t = m.find(gte("field", 5));
mongoClient.close();
```

Listing 1.2: Typical code for an HBase filter query.

```java
Configuration c = HBaseConfiguration.create();
Connection connection = ConnectionFactory.createConnection(c);
Table table = connection.getTable(TableName.valueOf("test"));
Scan scan = new Scan();
RowFilter gte = new RowFilter(GREATER_OR_EQUAL,
                new BinaryComparator(Bytes.toBytes(5)));
scan.setFilter(gte);
ResultScanner scanner = table.getScanner(scan);
table.close();
```

Motivated by the evident lack of a unified query language, in this paper, we propose NoDA, an abstraction layer in the form of an API for querying NoSQL stores with native support for geospatial operations. NoDA enables application developers to query different NoSQL stores using a single code base, in spite of the peculiarities of each NoSQL system's query language. Effectively, this allows the seamless transition of application code from one NoSQL store to another, without the need of substantial changes. NoDA is implemented in Java, thus can be used by any JVM-enabled language (e.g., Scala). It executes the query on the specified NoSQL store and can return the results as a Spark DataFrame. Essentially, this feature makes NoDA compatible with big data processing frameworks (such as Apache Spark), and allows "pushing-down" operators (such as filters, projections, aggregates) to the storage engine for increased efficiency. Standard spatio-temporal operations are

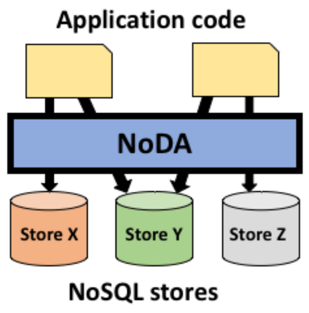

Figure 1: The NoDA abstraction layer.

currently supported in NoDA – including box range queries, circle range queries and *k*NN queries – as well as more general operators, such as aggregations and sorting.

In brief, the contributions of this work include:

- We propose NoDA, an abstraction that allows unified access to NoSQL stores, thus breaking the barrier to entry that developers typically face when querying big data.
- We demonstrate the applicability of NoDA in the context of spatio-temporal data, which is not naturally supported by all NoSQL stores, thus providing generic geospatial operations (range queries, *k*NN, and aggregations).
- We instantiate NoDA over two different NoSQL stores, a document-oriented store that supports geospatial operations (MongoDB), and a wide-column store (HBase) that is oblivious to spatial data, thereby showing the wide potential of our approach.

The remainder of this paper is structured as follows: Section 2 introduces the NoDA abstraction layer, including its instantiation over MongoDB and HBase. Section 3 reviews related work and Section 4 concludes our study and provides hints for future research directions.

2 THE NODA ABSTRACTION LAYER

In this Section, we provide a description of the NoDA abstraction (Section 2.1), followed by the details on how NoDA has been implemented over two popular NoSQL stores, namely a document-oriented NoSQL store (MongoDB, Section 2.2) and a wide-column store (HBase, Section 2.3).

2.1 NoDA Description

Figure 1 depicts the NoDA abstraction layer on top of NoSQL stores. NoDA resides between application code and data storage as a bridge for data access, and aims at "hiding" the query language of the underlying store from the developer. Essentially, a big data developer expresses her code using the NoDA abstraction, and this hides the peculiarities and complexity of accessing the specific NoSQL store. In this way, the exact same code can be used to query data stored in different NoSQL stores, no matter how different their query languages are.

The objective of NoDA is to offer a developer-friendly abstraction, which can be exploited to provide *simple* and *unified* access to scalable NoSQL stores. *Simple* in terms of using a familiar vocabulary of generic operations (filter, project, sort, etc.), without mixing the data model and the query language of the individual NoSQL store in the application code. *Unified* because the exact same operations are used for querying different NoSQL stores.

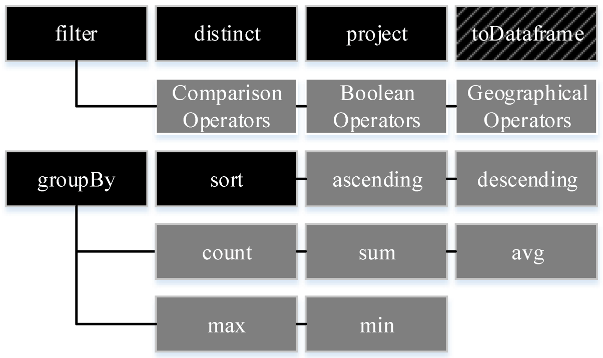

Figure 2: Operators supported in NoDA.

2.1.1 Functionality.

Figure 2 provides an illustrative overview of the operators offered by NoDA. *Primitive operators* include *filter*, *project*, *sort*, *groupby* and are depicted in black boxes. Grey boxes indicate a set of operators provided for defining the functional behaviour of primitive operations. For example, different types of filtering are supported: based on equality, on comparison, or on satisfying a geographical predicate.

Listing 2.1: Set up a connection to a NoSQL database by combining it with a Spark Session.

```
1  NoSqlDbSystem.initialize();
2
3  SparkSession session = SparkSession.builder().master("local")
4  .appName("MongoSparkConnectorIntro").getOrCreate();
5
6  //YYY method is the name of the NoSQL database in which we
7  //connect. It can be MongoDB or HBase.
8  NoSqlDbSystem noSqlDbSystem = NoSqlDbSystem.YYY()
9  .host("192.168.1.1").database("test").username("user")
10 .password("pass").port(28017).sparkSession(session).build();
11
12 NoSqlDbOperators noSqlDbOp = noSqlDbSystem
13                       .operateOn("table_name");
14
15 //doing data query operations by utilizing noSqlDbOp
16
17 noSqlDbSystem.closeConnection();
18 session.close();
```

Listing 2.1 shows the interface of NoDA used for setting up a connection to a NoSQL database. In lines 8-10 the connection is established by passing the necessary parameters (IP, port, authentication details, database name), and it is associated with a Spark session (line 3). This is not mandatory, but it is useful for certain complex data processing tasks (e.g., joins) that cannot be "pushed-down" to the NoSQL store, thus NoDA can populate Spark's DataFrames with data. Then, a *NoSqlDbSystem* object is initialized on a specific table (or its equivalent in NoSQL terms), which can be used for querying the NoSQL store. It should be emphasized that the code is oblivious to the exact NoSQL store used, apart from the *YYY* method (line 8). For example, to connect to a MongoDB instance, we would simply set *YYY=MongoDB*.

Listing 2.2: Definition and Execution Phase of Primitives.

```
Dataset<Row> dataset = noSqlDbOp
    .filter( ... ).filter( ... ) //definition phase
    .groupBy( ... ).sort( ... ) //definition phase
    .project( ... ) //definition phase
    .toDataframe(); //execution phase
```

Listing 2.2 depicts the template code needed for expressing a query in NoDA. A query can be expressed by applying a sequence of primitive operators on a *NoSqlDbSystem* object. The same syntax is used to express simple and complex queries. For example, a simple query such as projecting a field can be expressed using a single primitive operator (*project*), whereas a complex operation can be expressed by combing multiple operations (e.g., two *filter* operators followed by a *sort* operator).

At a more technical level, query primitives in NoDA are separated in two phases: *definition* and *execution* phase. The query primitives correspond to stages in a multi-stage pipeline. Multiple stages can be declared in the pipeline, which are then executed sequentially, when an operator is found that corresponds to the execution phase. As shown in the example of Listing 2.2, most primitive operators belong to the definition phase, whereas the *toDataFrame* operator belongs to the execution phase.

2.1.2 Spatial and Spatio-temporal Operators. An important feature of NoDA is that it was designed to support spatial and spatio-temporal operations. Note that these query types are not supported by all NoSQL stores, thus making it more challenging to provide this functionality in the first place.

Towards this goal, in addition to common operators such as boolean and comparison ones, NoDA supports operators oriented to spatial (2D) and spatio-temporal (3D) data, called *Geographical Operators* (or Geo-Operators in short). The offered Geographical Operators are: inGeoPolygon, inGeoBox, inGeoCircleKm, inGeoCircleMeters, inGeoMiles, nearestNeighbors and inGeoTemporalBox.

Listing 2.3: Spatial rectangle query and spatial circle query

```
// Spatial rectangle query
Coordinates c1 = Coordinates.newCoordinates(23.6266, 37.9262);
Coordinates c2 = Coordinates.newCoordinates(23.6682, 37.9477);
int count = noSqlDbOp.filter(inGeoBox("location", c1, c2))
            .count();

// Spatial circle query whose radius is 2Km.
Coordinates c = Coordinates.newCoordinates(23.7613, 37.9864);
int count = noSqlDbOp.filter(inGeoCircleKm("location", c, 2))
            .count();
```

Listing 2.3 shows examples of a box range query and a circular range query expressed in NoDA. Passing a geographical operator type to the *filter* primitive is equivalent to a filtering operation on a specific type of spatial data. In this way, the developer can retrieve data using spatial constraints by passing Geo-Operators as arguments to primitive operations. Nearest neighbor queries are very important for spatial data, therefore they are supported by NoDA as shown in Listing 2.4. Finally, NoDA supports also spatio-temporal data, as shown in Listing 2.5, where a spatio-temporal range query is expressed.

Listing 2.4: Spatial k-NN query.

```
Coordinates c = Coordinates.newCoordinates(23.7613, 37.9864);
int count = noSqlDbOp
            .filter(nearestNeighbors("location", c, 2))
            .toDataframe();
```

Listing 2.5: Spatio-Temporal Box query.

```
SimpleDateFormat s =
    new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS");
Date d1 = s.parse("2017-12-01T00:00:00.000Z");
Date d2 = s.parse("2017-12-02T23:59:59.999Z");
Coordinates c1 = Coordinates.newCoordinates(23.6266, 37.9262);
Coordinates c2 = Coordinates.newCoordinates(23.6682, 37.9477);
int count = noSqlDbOp.filter(
            inGeoTemporalBox("location", d1, d2, c1, c2))
            .count();
```

All these examples demonstrate the simplicity of NoDA, which abstracts away the individual query language of the underlying NoSQL store. In fact, all above examples are used to query *any* NoSQL store, without any changes, since the only place where we declare the specific NoSQL store used is in the initialization of the *NoSqlDbSystem* object.

2.2 NoDA over MongoDB

MongoDB is a popular document-oriented NoSQL store, where data is modeled as documents. In order to implement NoDA over MongoDB, we represent each spatial or spatio-temporal position as a document. Obviously, if the record contains additional attributes (e.g., speed, acceleration, etc.), these are also represented as fields of the same document. Also, MongoDB provides a spatial extension, which is based on GeoJSON representation, and spatial indexing as well as spatial operators are supported. This makes the internal implementation of NoDA over MongoDB easier.

MongoDB provides spatial operators, such as *$geoNear*, *$geoWithin*, *$geometry*, *$centerSphere*. In the implementation of the Geo-Operators of NoDA over MongoDB, we translate the query parametes (e.g., point coordinates or query radius) to these spatial operators. For example, the operators inGeoBox and inGeoPolygon use the *$geoWithin* with the *$geometry* query operator of MongoDB, selecting documents with geospatial data enclosed in the specified geometry. The inGeoCircle operator uses the *$geoWithin* with the *$centerSphere* query operator of MongoDB, selecting documents within the bounds of the circle. The nearestNeighbors operator exploits the *$geoNear* operator of MongoDB.

The efficiency of the above operators is supported by built-in spatial indexes of MongoDB (2d and 2dSphere). Also, Compound indexes can be used to index spatial and temporal data jointly. In this case, NoDA can exploit the spatial indexing provided by MongoDB for efficient access. However, we found that some spatial operations are not supported as efficiently as possible. We currently investigate on methods to improve the performance of such operations, e.g., nearest neighbor queries.

2.3 NoDA over HBase

Supporting the NoDA abstraction over another NoSQL store raises other challenges. For example, HBase is a wide-column store that follows the idea of BigTable [2], which does not support spatial data natively. This is also the case for several other NoSQL stores that rely on key-value pairs, and mainly support key-based operations, including range scans.

The solution to the problem is mapping the spatial (or spatio-temporal) data to 1D values, which can then be used as keys in HBase. Locality-preserving 1D mappings have been extensively used in spatial databases (e.g., using space-filling curves), but also in NoSQL context (e.g., MD-HBase [6], QUILTS [7], etc.). NoDA follows the same principle in order to become applicable over key-based NoSQL stores.

In more detail, given a spatio-temporal record $\{x, y, t\}$, we derive a 1D mapping using Z-order, which is then used as key for inserting the record in HBase. Then, spatio-temporal queries such as range or kNN can be equivalently expressed as range searches over the 1D values (e.g., [9]). The NoDA instantiation for HBase implements this functionality, thus hiding this complexity from the developer, while still exposing the same interface, as described above.

3 RELATED WORK

NoDA is related to the storage layer of platforms that provide a holistic solution for big spatial data management and analytics through application interfaces.

Many of these platforms operate on a particular type of data, like UlTraMan [4], which is oriented towards trajectory data. UlTraMan supports querying by trajectory ID, range queries, kNN queries, and co-movement pattern mining. Furthermore, UlTraMan requires the data to be moved in its storage layer; during this operation, a pre-processing step occurs, which transforms the data and partitions them into the available nodes of the cluster. NoDA does not require the data to be moved; it queries the data on their original NoSQL store. Similarly, DITA [8] is a distributed in-memory trajectory analytics system on Spark, providing an interface for trajectory analysis. However, DITA mainly targets efficient in-memory processing and analysis of spatial data. DITA could benefit from coupling with NoDA as underlying layer for accessing NoSQL storage.

Pyro [5] is another system for querying spatio-temporal data. Pyro builds an index on top of HBase, which is used to generate the set of range scans needed for a spatial query. Then, it evaluates the query by executing a multi-scan phase, which processes all produced range scans together. Pyro is designed to utilize the benefits of HBase, and cannot be trivially ported to other NoSQL stores.

GeoMesa [1] is a suite of tools, with similar objectives to NoDA abstraction layer. It supports spatial queries on column-based and key-value based NoSQL stores, by using the Common Query Language (CQL). GeoMesa builds indexes, by exploiting 1D mappings using Z-order, on top of the processing layer, to enable fast retrieve of the query result set. This resembles our approach for instantiating NoDA over HBase. However, a critical difference is that NoDA exposes specific data operators to the developer, thus aiming at easy integration with application code.

NoSQL systems supporting multiple data models, while persisting the same query language, are also related to NoDA. Such systems, also called polyglot, are surveyed in [3]. ArangoDB[2] integrates document, key-value and graph data models in a single system. It uses AQL as its query language which also supports geospatial functions for querying spatial data. OrientDB[3] combines document, key-value, reactive and object-oriented models in a single system. OrientDB's query language is based on SQL and supports geospatial queries, along with some extensions for manipulating trees and graphs. In contrast to these systems, NoDA is designed to be used *on top of* any existing NoSQL store.

4 CONCLUSIONS

In this paper, we introduce NoDA, an abstraction layer for NoSQL stores that allows unified access to heretogeneous data models and storage systems. Its primary focus is on scalable mobility data management, in particular spatial and spatio-temporal operators. NoDA has been implemented on top of MongoDB and HBase, thus demonstrating the NoDA concept in practice.

Many directions for future work are opened up. First, we intend to provide a declarative language on top of NoDA. Another direction is to extend NoDA to support other types of NoSQL stores, such as graph databases. Also, we intend to enrich the supported queries for mobility data towards trajectory operators. Last, but not least, we are going to explore techniques for more efficient support for spatio-temporal operators, since our preliminary experiences show that NoSQL are not optimized for mobility data.

ACKNOWLEDGMENTS

This research work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780754 (Track&Know project), and from the Hellenic Foundation for Research and Innovation (HFRI) and the General Secretariat for Research and Technology (GSRT), under grant agreement No 1667.

REFERENCES

[1] Rick Cattell. 2010. Scalable SQL and NoSQL data stores. *SIGMOD Record* 39, 4 (2010), 12–27.
[2] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2008. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.* 26, 2 (2008), 4:1–4:26.
[3] Ali Davoudian, Liu Chen, and Mengchi Liu. 2018. A Survey on NoSQL Stores. *ACM Comput. Surv.* 51, 2 (2018), 40:1–40:43.
[4] Xin Ding, Lu Chen, Yunjun Gao, Christian S. Jensen, and Hujun Bao. 2018. UlTraMan: A Unified Platform for Big Trajectory Data Management and Analytics. *PVLDB* 11, 7 (2018), 787–799.
[5] Shen Li, Shaohan Hu, Raghu K. Ganti, Mudhakar Srivatsa, and Tarek F. Abdelzaher. 2015. Pyro: A Spatial-Temporal Big-Data Storage System. In *2015 USENIX Annual Technical Conference, USENIX ATC '15, July 8-10, Santa Clara, CA, USA.* 97–109.
[6] Shoji Nishimura, Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. 2011. MD-HBase: A Scalable Multi-dimensional Data Infrastructure for Location Aware Services. In *12th IEEE International Conference on Mobile Data Management, MDM 2011, Luleå, Sweden, June 6-9, 2011, Volume 1.* 7–16.
[7] Shoji Nishimura and Haruo Yokota. 2017. QUILTS: Multidimensional Data Partitioning Framework Based on Query-Aware and Skew-Tolerant Space-Filling Curves. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017.* 1525–1537.
[8] Zeyuan Shang, Guoliang Li, and Zhifeng Bao. 2018. DITA: Distributed In-Memory Trajectory Analytics. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018.* 725–740.
[9] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. 2010. Efficient and accurate nearest neighbor and closest pair search in high-dimensional space. *ACM Trans. Database Syst.* 35, 3 (2010), 20:1–20:46.

[1] https://www.geomesa.org/
[2] https://www.arangodb.com/
[3] https://orientdb.com/