BBoxDB Streams: Distributed Processing of Real-World Streams of Position Data

Jan Kristof Nidzwetzki
FernUniversität Hagen
Hagen, Germany
jan.nidzwetzki@studium.fernuni-hagen.de

Ralf Hartmut Güting
FernUniversität Hagen
Hagen, Germany
rhg@fernuni-hagen.de

ABSTRACT

BBoxDB Streams is an extension of the key-bounding-box-value store BBoxDB. The extension allows the handling of multi-dimensional data streams. Multi-dimensional streams consist of *n*-dimensional elements, such as position data (e.g., two-dimensional positions of cars or three-dimensional positions of aircraft). In this demonstration, we show how BBoxDB Streams can be used to process data streams of position data in a distributed manner. The software allows the user to capture data streams and process continuous queries. Continuous range queries or continuous spatial joins are supported. The GUI of BBoxDB Streams shows the query results interactively as an overlay over a map. For the demonstration, public real-world data streams with positions of aircraft and transport data are processed. Continuous range queries such as *which aircraft is currently in the area of Berlin?* or continuous spatial join queries such as *which bus drives currently through a forest?* are executed, and the results can be observed in real-time. The spatial joins are executed between the stream data and previously stored static geographical information (e.g., the polygons of roads or forests), which are fetched from the OpenStreetMap Project.

1 INTRODUCTION

Data streams consisting of position data are ubiquitous. For example, aircraft periodically broadcast their positions via *ADS-B messages* (*Automatic Dependent Surveillance–Broadcast*), and the positions of buses, trains, or ferries of public transport companies are available in real-time via the internet in GTFS format (*General Transit Feed Specification*). Processing streams containing position data is an essential topic in location-aware applications. The ability to capture data streams and the near real-time execution of queries is required to deal with the information of the stream. Data streams can contain a lot of elements, and queries can be expensive to evaluate. Therefore, a scalable solution is required to process data streams.

BBoxDB Streams is an extension of *BBoxDB* [15], which allows the efficient handling of *n*-dimensional data streams. BBoxDB streams implements a novel way to execute efficient continuous joins between dynamic elements from a data stream and static already stored *n*-dimensional big data. This capability is shown for the first time in this demonstration. Besides, the GUI of BBoxDB was enhanced to execute queries on data streams and show the results interactively. This new enhancement is shown in this demonstration for the first time. BBoxDB streams is included in BBoxDB since version 0.9.5 and licensed under the Apache 2.0 license. The software can be freely downloaded from the website of the project [3].

In this demonstration, we show two types of queries with BBoxDB Streams: (1) *continuous range queries*, and (2) *continuous spatial joins*. Continuous range queries can answer questions such as *which aircraft are inside of a specific region of the airspace?*. Continuous spatial joins can be used to join the dynamic position data of the stream with static data (e.g., geographical information). With a continuous spatial join, queries such as *which buses are closer than 10 miles to a forest?* or *which bus drives on a particular street?* can be answered.

The rest of the paper is organized as follows: Section 2 describes the key-features of BBoxDB and BBoxDB Streams. Section 3 describes our demonstration. Section 4 describes the related work. Section 5 concludes the paper.

2 BBOXDB AND BBOXDB STREAMS

The amount of data is increasingly growing. NoSQL databases like *distributed key-value stores* (DKVS) are often used to handle large amounts of data. In a DKVS, the data are assigned to the nodes of a cluster. Each node stores only a part of the whole dataset. A value is stored under a given key. Using this key, the value can be retrieved. The key is the access path to the data. A key for a one-dimensional value can be easily chosen. For data with a higher dimensionality, a key is hard to choose. Which key should be used for the geographic information about a road? Using the name of the road as the key does not help to access the data when a spatial range query is performed; the name does not contain the information where the road is located. To answer such range queries, a full data scan has to be performed; all stored tuples have to be loaded and it needs to be tested whether or not the stored value intersects with the given query rectangle. This is an expensive operation that is performed on all nodes of the distributed system.

BBoxDB was designed to solve this problem. BBoxDB is a distributed *key-bounding-box-value store* (KBVS) which supports the efficient storage and retrieval of *n*-dimensional data.

2.1 Basic Concepts of BBoxDB

BBoxDB is a distributed generic datastore optimized for the handling of *n*-dimensional big data. Values are stored as byte arrays together with a key and an *n*-dimensional bounding box as tuples. The bounding box describes the location of the *tuple* in the *n*-dimensional space. Point and non-point data are supported by BBoxDB. Tuples are grouped together in *tables* and multiple tables of the same dimensionality can be stored together in a *distribution group*. The space is split automatically to ensure almost equal-sized partitions; the data of these partitions (called *distribution regions*) are assigned to the nodes of a cluster. The tables of the same distribution group are distributed in the same way; this means the tables are stored co-partitioned, which enables the execution of efficient spatial joins. No data need to be transferred through the network; all join partners are stored on

the same node. The complete system is highly available, data can be replicated and failing nodes are handled automatically.

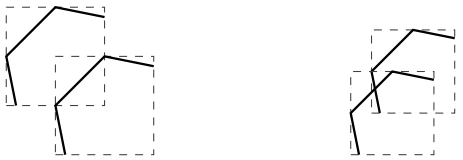
Operations: Data are stored in BBoxDB with the `put(table, key, hrect, value)` operation. As `hrect` parameter, an n-dimensional bounding box (a hyperrectangle) has to be specified. One-dimensional point data (as used in DKVS) can also be stored in BBoxDB. In this case, the bounding box degenerates to a point in the one-dimensional space.

Data are retrieved by the `queryByRect(table, hrect)` operation, which retrieves all tuples whose bounding box intersects with the query bounding box. The operation `join(table1, table2, hrect)` executes a spatial join between the two tables in the specified region in space.

Indexing: BBoxDB uses a two-level indexing structure that enables the efficient execution of range queries. The *global index* is used to map the distribution regions to the nodes of the cluster ($distribution\ region \rightarrow \mathcal{P}(nodes)$)[1]. The space is partitioned (split and merged) by a *space partitioner* automatically, based on the stored data [13]. Splitting and merging the space is done transparently in the background without interrupting the access to the data. BBoxDB provides multiple algorithms for the global index. The used algorithm can be specified (KD-Trees [4] or Quad-Trees [5]) when the distribution group is created. The *local index* is stored on the nodes and maps from the space to the stored tuples ($space \rightarrow tuples$). This index is implemented by an R-Tree [7] which is stored on the nodes.

User-defined filters: BBoxDB is a generic data store; the stored values are a plain array of bytes. BBoxDB does not understand the semantics of the stored data. Therefore, operations are executed primarily on the bounding boxes of the tuples. *User-defined filters* [14] (UDFs) can be used to decode the bytes of a value (e.g., GeoJSON encoded data) and to refine the bounding box based operations of the query processor. UDFs are developed by the user of the system. Only the user who has stored the data knows how to interpret the values of the data. Besides, BBoxDB ships with a collection of UDFs for common data formats. UDFs are written in Java and can use existing libraries.

One of the UDFs that have been included is capable of decoding GeoJSON data. Using this UDF, a bounding box based spatial join is refined to a spatial join on the real geometries of the values. Intersecting bounding boxes is a necessary but not sufficient criterion for a spatial join (see Figure 1).



(a) Two non-intersecting spatial objects. (b) Two intersecting spatial objects.

Figure 1: Two spatial objects (solid line) with intersecting bounding boxes (dashed line). In (a), the spatial objects do not intersect, while in (b), the spatial objects do intersect.

This UDF is used in the demonstration to refine the continuous spatial joins. The property map of GeoJSON encoded objects (see Listing 1) can also be taken into consideration in the UDF. For example, this can be used to filter streets of a specific name like *Elizabeth Street*. Queries such as *find all buses which are on a street named Elizabeth Street* become possible.

[1]When replication is used, one distribution region is mapped to multiple nodes.

2.2 BBoxDB Streams

BBoxDB is an extension of BBoxDB that allows the handling of data streams. With the extension, data streams can be captured and continuous queries can be executed. Figure 2 shows the architecture of the extension. The upper part of the figure shows the stream capturing part, the lower part the query processing part.

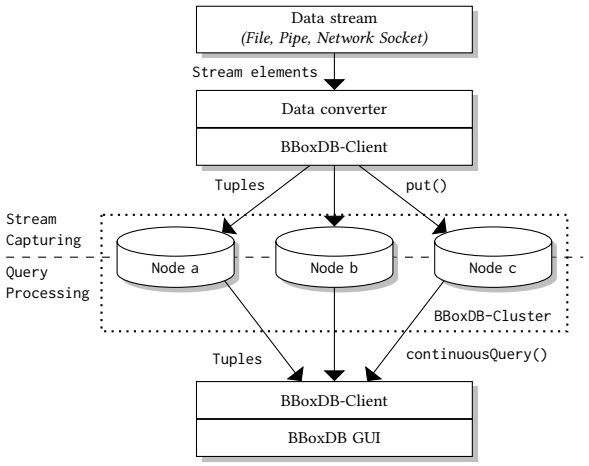


Figure 2: Handling a data stream with BBoxDB. The data stream is captured, converted into tuples, and written to the BBoxDB cluster. Afterward, the continuous queries are executed and the result can be consumed.

Stream Capturing: BBoxDB Streams captures data continuously from an input source like a *file*, a *pipe*, or a *network socket*. After a stream element is read, the element is converted into a BBoxDB tuple and sent to the BBoxDB nodes of the cluster[2]. To communicate with the BBoxDB nodes, the regular BBoxDB-Client library is used. The library manages the connection to the nodes, observes the global index, and executes the operations on the necessary nodes. Changes of the global index or the available nodes are handled. The converted stream elements are written to the BBoxDB cluster by executing the `put()` operation. The bounding box of these tuples is compared with the global index; the tuple is written to all nodes that are responsible for the region is space. On these nodes, the potential join partners are stored for a spatial join; the stream elements become co-partitioned to the already stored data, and efficient spatial joins become possible. After a node receives a tuple, the registered continuous queries are executed. The highly-available architecture is also used for the processing of the streams. Distribution groups can be stored replicated, in this case the continuous queries are also registered on multiple nodes. As the stream is written to a table, BBoxDB splits the space, updates the global index, and re-distributes uneven distributed tables automatically as described in [15, p. 20].

Query Processing: BBoxDB Streams enhances BBoxDB by two operations for the handling of continuous queries: (1) `continuousQuery(queryPlan)` and (2) `cancelQuery(id)`. The first operation registers a new query while the second operation cancels a previously registered query. The existing BBoxDB-Client

[2]BBoxDB ships with some data converter for common data formats like GTFS or ADS-B. The converter decodes the input data, calculates the bounding box and creates a tuple. Further data converters can be added easily by a user. For certain stream types, multiple elements from the stream are combined into one BBoxDB tuple. For example, the ADS-B format defines multiple message types. Three different message types have to be read to get the current data of an aircraft.

was enhanced by BBoxDB Streams to register continuous queries. The BBoxDB-Client automatically registers the queries on the required nodes of the cluster; this part of the client library was re-used. In this demonstration, the GUI of BBoxDB uses the BBoxDB-Client to register the queries and to obtain the results. These tuples are consumed by the GUI and shown as an overlay of a map. The stream handling functionality is now integrated into the regular client library. Any application can create queries on data streams and consume the results.

The query plan defines the operations of the continuous query. In the query plan, (1) the *type* of the query (range query or spatial join), (2) *transformations*, and (3) *filters* are defined. Technically, the query plan is a JSON document that is sent to the BBoxDB nodes. A helper class is available, which allows the easy and syntactically correct creation of these query plans.

Transformations allow the modification of the stream elements and the potential join partners. For example, the bounding box of an aircraft can be extended and joined with the obstacles in the airspace. Due to the enhancement of the bounding box, a possible collision is detected and reported before the aircraft and the obstacle actually collide. Also the potential collision of two aircraft can be detected. This is done by storing the data stream in a table and performing a continuous spatial join between the new stream elements and the materialized stream elements from the table.

Filters allow removing elements from the stream or from the list of potential join partners. For example, process only the aircraft with a call sign starting with `LH`). In addition, UDFs can also be used as a filter.

The complete architecture of BBoxDB and BBoxDB streams is horizontally scalable. When more static data have to be stored, the existing distribution regions can be split, and these new regions can be assigned to further BBoxDB nodes. This can also be done to process a larger stream or more continuous queries. The use of more distribution regions splits up the stream into more parts. Each part is handled by an individual node that has its own resources to capture the stream and execute the registered queries.

3 DEMONSTRATION

A cluster of five nodes is used in our demonstration, which is located at our university. Each node contains an Intel Xeon E5-2630 CPU, 32 GB of memory and four 1 TB hard disks. All the nodes are connected via a 1 Gbit/s network and running Java 8 on a 64 bit Ubuntu Linux. A notebook is used to run the GUI and to perform the demonstration.

3.1 Data Streams for the Demonstration

In this demonstration, two real-world data streams are used: (1) *The ADS-B aircraft data stream* and (2) *the Sydney transport data stream.*

An aircraft continuously broadcasts its position periodically via radio as ADS-B transmissions. The transmissions contain the position of the aircraft, the height, the call sign, and some more information. However, an ADS-B receiver captures only the transmissions in a radius of some miles around the antenna. Websites such as adsbhub.org [19] provide a service to aggregate the feeds of several individual stations into a global feed, containing the flight data of the whole world.

The government of the state of New South Wales in Australia operates the *NSW open data portal* [17]. On this portal, real-time data about buses, ferries, metros, and trains of the region are published. A GTFS encoded real-time feed of the data can be subscribed.

For our demonstration, the elements of both data streams are used and converted into GeoJSON objects. GeoJSON is a format that can be read and understand by a human (in contrast to binary-encoded GTFS data), which makes it suitable for demonstration purposes. Listing 1 contains one element from the public transport data steam after it is converted into GeoJSON. In addition to the position of the bus, further *properties* are contained which contain additional information such as the route or the speed of the vehicle.

Listing 1: Bus trip data converted into GeoJSON

```
{
 "geometry":{
  "coordinates":[151.17762756347, -33.92598342895],
  "type":"Point"
 },
 "type":"Feature",
 "properties":{
  "Speed":"19.2",
  "TripStartDate":"20200121",
  "TripScheduleRelationship":"SCHEDULED",
  "OccupancyStatus":"MANY_SEATS_AVAILABLE",
  "TripStartTime":"02:00:00",
  "RouteID":"2437_N20",
  "Timestamp":"1579530867",
  "TripID":"883447",
  "Bearing":"77.0",
 }
}
```

For executing spatial joins, we fetched the *planet* data set from the *OpenStreetMap Project* [18], converted this data set into GeoJSON, and stored it in the BBoxDB cluster. The dataset contains the geographical information of the whole world. We imported the *roads* (146 060 493 elements - 67 GB) and the *forests* (5 187 592 elements - 5.4 GB) in our BBoxDB cluster for the demonstration.

3.2 The GUI of BBoxDB

The GUI of BBoxDB shows information about the BBoxDB cluster, the data distribution, and can be used to perform queries. BBoxDB Streams enhances the GUI in such a way that continuous queries are supported. The GUI is optimized for the handling of GeoJSON encoded data. On the main screen, a map of the world (fetched dynamically from the OpenStreetMap Project) is shown. The mouse can be used to create a query rectangle, and a window that is automatically opened allows one to specify the desired query. Continuous range and continuous spatial joins can be executed, and user-defined filters can be applied.

The geometries of the result tuples are shown as an overlay over the map. In addition to the location, the stream elements contain further information. Placing the mouse cursor over an element opens a tooltip. The tooltip contains all the additional information that is contained in the GeoJSON object (e.g., the height of an aircraft or the trip id of a bus). The area of the GUI below the map shows details about the used cluster (i.e., IP, software version, available disks, disk space, CPUs).

Different practical queries can be formulated and observed in the GUI. Figure 3 shows the visualization of an ADS-B data stream of aircraft in the region of Berlin, Germany. Another example (see Figure 4) shows the visualization of a GTFS data stream of metro buses in Sydney, Australia. Additional operations, such as spatial joins between forests and buses (*Which bus drives currently through a forest*?) or roads and buses (*Which buses are driving on the Elizabeth Street in Sydney?*) can be performed and displayed in the GUI.

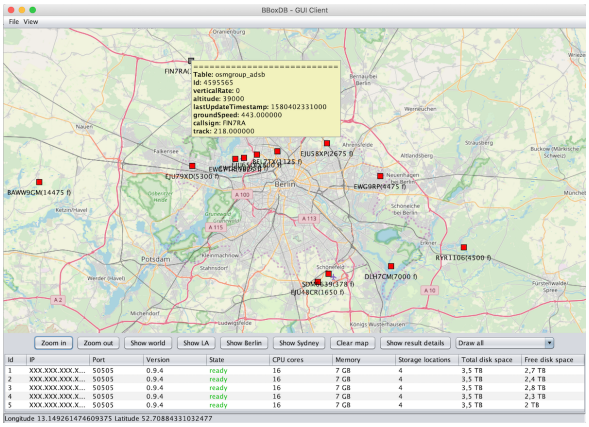


Figure 3: Observing aircraft traffic over Berlin, Germany.

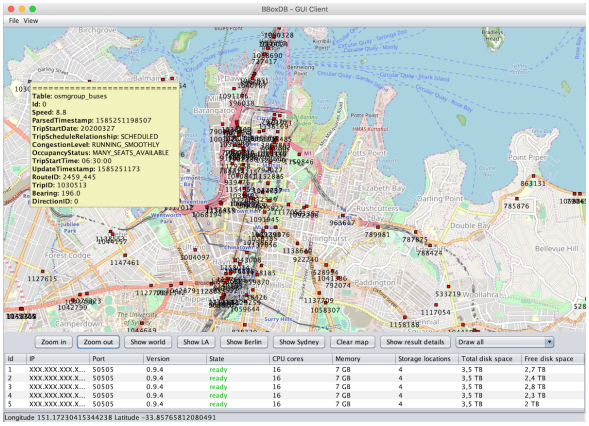


Figure 4: Observing bus traffic in Sydney, Australia.

4 RELATED WORK

BBoxDB and BBoxDB streams have related work in the area of key-value stores and stream processing systems.

Key-Value Stores: During the last decade, NoSQL databases have become popular. They omit features from RDBMS, such as transactions and permanent consistency. This allows NoSQL systems to scale better horizontally. Distributed key-value stores such as *Cassandra* [9] or *HBase* [1] provide simple methods to manage large amounts of key-value pairs. These are optimized for one-dimensional data, since handling n-dimensional data is a laborious task in such systems (see Section 2).

KVS with support for n-dimensional data: *MD-HBase* [16] is a multi-dimensional extension of HBase that allows the efficient storage and retrieval of multi-dimensional data. MD-HBase employs *Quad-Trees* and *K-D Trees* together with a Z-Curve to build an index. Systems such as *EDMI - Efficient Distributed Multi-dimensional Index* [21], *Pyro* [10], and *HGrid* [8] are also enhancements of HBase which use an additional index layer to store multi-dimensional data in HBase. However, operations such as spatial joins or continuous queries are not supported by these systems, and these systems only support point data.

Stream Processing Systems: *Apache Flink* [6], *Apache Spark Streams* [11], *Apache Storm* [2], and *Apache Kafka* [12] are widespread stream processing systems. These systems are not optimized to compare the stream data with larger previously-stored datasets. Operations like geometric indexing or spatial joins are not supported by these systems. In [20] an extension of Apache Storm for the handling of spatial data streams is proposed. However, the paper focuses only on 2-dimensional point data; BBoxDB streams can handle n-dimensional point and non-point data.

5 CONCLUSION

In this demonstration, we have shown the capabilities of BBoxDB Streams for the first time. Two real-world data streams are captured and processed. Queries such as continuous range queries and continuous spatial joins are performed on these streams. The spatial joins are executed between the dynamic data from the data stream and static stored data fetched from the OpenStreetMap Project. The results of the queries are visualized using an enhanced version of the GUI of BBoxDB and user-defined filters are used to refine the bounding box based operations of the query processor. In this demonstration, many aspects of the architecture of the system are only superficially addressed. Topics such as the integration with BBoxDB, the scalability, the filters, and transformations have to be described more precisely. We plan to discuss these topics in detail together with an experimental evaluation of BBoxDB Streams in a full research paper.

REFERENCES

[1] Apache HBase 2020. Website of Apache HBase. https://hbase.apache.org/ [Online; accessed 03-Nov-2020].

[2] Apache Storm 2021. Website of the Apache Storm project. https://storm.apache.org/ - [Online; accessed 20-Jan-2021].

[3] BBoxDB 2018. Website of the BBoxDB project. http://bboxdb.org [Online; accessed 03-Nov-2020].

[4] J. L. Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM* 18, 9 (Sept. 1975), 509–517.

[5] R. A. Finkel and J. L. Bentley. 1974. Quad Trees a Data Structure for Retrieval on Composite Keys. *Acta Inf.* 4, 1 (March 1974), 1–9.

[6] E. Friedman and K. Tzoumas. 2016. *Introduction to Apache Flink: Stream Processing for Real Time and Beyond* (1st ed.). O'Reilly Media, Inc.

[7] A. Guttman. 1984. R-trees: A Dynamic Index Structure for Spatial Searching. *SIGMOD Rec.* 14, 2 (June 1984), 47–57.

[8] D. Han and E. Stroulia. 2013. HGrid: A Data Model for Large Geospatial Data Sets in HBase. 910–917.

[9] A. Lakshman and P. Malik. 2010. Cassandra: A Decentralized Structured Storage System. *SIGOPS Oper. Syst. Rev.* 44, 2 (April 2010), 35–40.

[10] S. Li, S. Hu, R.K. Ganti, M. Srivatsa, and T.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.

[11] Z. Nabi. 2016. *Pro Spark Streaming: The Zen of Real-Time Analytics Using Apache Spark* (1st ed.). Apress, Berkely, CA, USA.

[12] N. Narkhede, G. Shapira, and T. Palino. 2017. *Kafka: The Definitive Guide Real-Time Data and Stream Processing at Scale* (1st ed.). O'Reilly Media, Inc.

[13] J.K. Nidzwetzki and R.H. Güting. 2018. BBoxDB - A Scalable Data Store for Multi-Dimensional Big Data (Demo-Paper). In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (Torino, Italy) *(CIKM '18)*. ACM, 1867–1870.

[14] J.K. Nidzwetzki and R.H. Güting. 2019. Demo Paper: Large Scale Spatial Data Processing With User Defined Filters In BBoxDB. In *2019 IEEE International Conference on Big Data (Big Data)*. 4125–4128.

[15] J.K. Nidzwetzki and R.H. Güting. 2020. BBoxDB: A Distributed and Highly Available Key-Bounding-Box-Value Store. *Distributed and Parallel Databases* 38 (June 2020), 439—-493.

[16] S. Nishimura, S. Das, D. Agrawal, and A. E. Abbadi. 2011. MD-HBase: A Scalable Multi-dimensional Data Infrastructure for Location Aware Services. In *Proceedings of the 2011 IEEE 12th International Conference on Mobile Data Management - Volume 01 (MDM '11)*. IEEE Computer Society, 7–16.

[17] Open Data Hub 2020. The Open Data Hub for New South Wales transport data. https://opendata.transport.nsw.gov.au - [Online; accessed 03-Nov-2020].

[18] OpenStreetMap Project 2020. Website of the OpenStreetMap Project. http://www.openstreetmap.org - [Online; accessed 03-Nov-2018].

[19] Website of adsbhub.org 2020. The Website of the adsbhub.org project. http://adsbhub.org - [Online; accessed 03-Nov-2020].

[20] F. Zhang, Y. Zheng, D. Xu, Z. Du, Y. Wang, R. Liu, and X. Ye. 2016. Real-Time Spatial Queries for Moving Objects Using Storm Topology. *ISPRS International Journal of Geo-Information* 5, 10 (2016).

[21] X. Zhou, X. Zhang, Y. Wang, R. Li, and S. Wang. 2013. Efficient Distributed Multi-dimensional Index for Big Data Management. In *Proceedings of the 14th International Conference on Web-Age Information Management* (Beidaihe, China) *(WAIM'13)*. Springer-Verlag, Berlin, Heidelberg, 130–141.