GIS Navigation Boosted by Column Stores

Foteini Alvanaki[2], Romulo Goncalves[1], Milena Ivanova[a 3], Martin Kersten[2], and Kostis Kyzirakos[2]

[1]NLeSC Amsterdam, The Netherlands {*r.goncalves@esciencecenter.nl*}

[2]Centrum Wiskunde & Informatica, Amsterdam, The Netherlands {{*f.alvanaki,martin.kersten,kostis.kyzirakos*}*@cwi.nl*}

[3]NuoDB Cambridge MA, USA {*mivanova@nuodb.com*}

ABSTRACT

Earth observation sciences, astronomy, and seismology have large data sets which have inherently rich spatial and geospatial information. In combination with large collections of semantically rich objects which have a large number of thematic properties, they form a new source of knowledge for urban planning, smart cities and natural resource management.

Modeling and storing these properties indicating the relationships between them is best handled in a relational database. Furthermore, the scalability requirements posed by the latest 26-attribute light detection and ranging (LIDAR) data sets are a challenge for file-based solutions.

In this demo we show how to query a 640 billion point data set using a column store enriched with GIS functionality. Through a lightweight and cache conscious secondary index called Imprints, spatial queries performance on a flat table storage is comparable to traditional file-based solutions. All the results are visualised in real time using QGIS.

1. INTRODUCTION

Point cloud data (or LIDAR) provide a wealth of information for various applications. In combination with auxiliary GIS data like cadastral data which contains information about the boundaries of properties, topological data that describe buildings, roads, rivers, lakes etc. and large collections of semantically rich objects which have a large number of properties, they form a new source of knowledge for urban planning, smart cities and natural resource management.

The production of point cloud data sets have increased in size over the past years to Tera byte scale due to its easy collection using airborne laser scanning. Airborne laser scanning is a remote sensing technology which is able to rapidly collect data at global scale. It collects large amounts of point data to be the base of digital surface or elevation models. The principle behind the measurements is as follows: the sensor emits a laser pulse through the terrain in a predefined direction and receives the reflected laser beam. Knowing the speed of light, the distance of the object can be calculated.

Such data is often combined with data from stationary laser scan devices which gather the ground-based structures. The de-facto standard to store and distribute the acquired data is the LAS [1]. The number of files and their density is increasing per scan base due to the sampling density of LIDAR sensors. From few million points per file, we soon will have billion points per file.

Storing and querying massive point cloud data is a challenging task. Just considering the number of properties, e.g. color, angle of scan, data etc., attached to each point gives a notion of the extent of the problem. The current version for LAS has a total of 23 properties excluding the X, Y, and Z coordinates.

The database community could not stay away from any attempt to efficiently manage massive point cloud data. The first approaches on storing and querying point cloud data already exist with the most known among them being the point cloud extension provided by Oracle and PostgreSQL. Both systems deviate from the established approach of representing geometries in a DBMS using a specialised geometry data type like the data type `POINT` as defined by the OpenGIS Simple Features Access standard [9]. Both systems base their performance on the physical reorganisation of data into blocks with each block being a condensed representation of multiple points. The point cloud tables contain then a number of blocks. This allows PostgreSQL and Oracle to reduce the space requirements. It also allows them to reduce the access times since locating a block that contains the data of interest (and possibly more) is faster when searching through blocks (less number of elements) than searching through each single point.

We deviate significantly from the aforementioned approaches and opt for a simple, yet efficient, storage model. Point cloud data is stored in a flat table where a different column is used for storing the X, Y, Z coordinates and the 23 properties of each point. Through a lightweight and cache conscious secondary index called Imprints [16] for a coarse filtering step followed by a regular grid for filtering refinement, it is possible to have the same functionality as a traditional spatial DBMS using spatial indexes.

During the demo session, the audience will see how our techniques offer the ground to design a new "spatially-enabled" DBMS. Inline with geospatial demonstration style, the DBMS is integrated with a visualisation tool for user interaction using pre-defined queries or user defined queries.

The remainder of the paper is organized as follows. In Section 2, we introduce column-stores, and file-based solutions and indexing strategies for point cloud data. In Section 3,

[a]All contributions were done while working at the NLeSC.

Proceedings of the VLDB Endowment, Vol. 8, No. 12

we describe our storage and query model. An overview of the demonstration is given in Section 4 followed by conclusions in Section 5.

2. BACKGROUND

In this section, we introduce both DBMS and file base approaches. For DBMS based approaches we identify the major differences between column-oriented and row-oriented architectures and why column-oriented architectures are more suitable for GIS systems. Furthermore, we explain how spatial information is represented and queried in the relational model context. For file based approaches, we revisit the techniques employed for indexing and physically re-organising the spatial data shared with traditional "spatially-enabled" DBMS.

2.1 Column-stores

In the recent years we have seen the introduction of a number of column-oriented database systems [10, 17]. For read-intensive analytical processing workload, such as the ones encountered in data warehouses, column-oriented architectures (column-stores) offer order-of-magnitude gains compared to traditional row-oriented architectures (row-stores).

2.1.1 MonetDB

MonetDB is a modern in-memory column-store database system, designed in the late 90's with a proven track record in various fields [12, 11] [1]. MonetDB has recently been extended with a novel lightweight secondary indexing scheme called column imprints [16], which we challenge in this demo.

A column imprint, or just imprints, is a collection of 64-bit vectors, each indexing data points that fit into a single cache line. Each of the 64 bits is associated with a range of values. A bit is set to 1 when the cache line indexed by the vector contains values in the corresponding range. The 64 ranges are global to an imprint and are decided based on the distribution of the values of the indexed column.

As described by the authors in [16], an imprint is used during query evaluation to limit data access, and thus minimise memory traffic. The compression for imprints is CPU friendly and exploits the empirical observation that data often exhibits local clustering or partial ordering as a side effect of the construction process. Most importantly, column imprint compression remains effective and robust even in the case of unclustered data, while other state-of-the-art solutions fail. The storage overhead, when experimenting with real world data sets, is just a few percent over the size of the columns being indexed.

We have selected to implement and test our approach in the context of MonetDB, not only because MonetDB is a mature representative of the column store family but mostly because it is an open source system that has been shown to perform well on similar scenarios [13]. Additionally, the operator-at-the-time paradigm followed by MonetDB provides an environment that allows for relatively easy development and incorporation of new operators that can benefit immediately from all the optimisation techniques present in MonetDB.

2.2 File-based solutions

File based solutions work directly with the standard LAS format and with one of its compression formats, Rapidlasso's LAZ [2] or ESRI ZLAS [3]. Such solutions are now encountering scalability problems. The new data acquisition technologies are providing high definition scans which increases the number of files to process and their size, and thus increase the number of data management challenges. For example, our test data *Actueel Hoogtebestand Nederland 2* (AHN2) [4], is stored and distributed in more than 60,000 LAZ files. It is already a large amount of files to be inspected for a simple selection by a robust file-based solution like Rapidlasso LAStools [5]. In [18], the authors for LAStools had to use a DBMS to store the metadata of each file in order to avoid the inspection of each file header, and run a *lassort* and *lasindex* to boost query performance. Such ETL process had the same cost as the data loading cost of a DBMS.

Furthermore, the complexity of ad-hoc queries cannot be expressed using a file-based solution, thus a declarative language for expressing such queries becomes a necessity. A declarative query language like SQL allows the user to easily express queries that combine numerous data sources like time-series, vector data, raster images etc. Hence, a front-end with a declarative language on top of a middle layer for workflow optimisation becomes also a requirement. Instead of reinventing the wheel, the extension of an existing DBMS is preferable when users want to combine, for example, vector geometries and point cloud data in their queries. RDBMS are also optimised for efficient IO strategies, i.e. scanning, pre-fetching, and cache management.

2.3 Indexing point cloud data

File-based solutions employ two indices for indexing point cloud data: space filling curves and octrees. A space filling curve is a curve whose range contains the entire 2-dimensional unit square. Space filling curves reduces the dimensionality of the data by mapping for example the X and Y coordinates in one dimension. Sorting the point cloud data using space filling curves is a common technique used by spatial DBMS and file-based solutions. A space filling curve is useful to exploit the spatial coherence of the data through spatial location codes [19].

In the case of Oracle spatial, the point cloud data type blocks can be sorted using a Hilbert space-filling curve [15]. Such spatial order is in most of the cases not exploited by the used compression techniques. An exception is PostgreSQL which uses compression techniques based on the spatial continuity of the blocks to offer better storage [18]. The same principles are followed by LAStools through *lassort* and *lasindex*.

3. ARCHITECTURE

In this section, we present the storage model that we chose for storing the point cloud data, how we loaded the full AHN2 dataset in MonetDB and how we process spatial selection queries over the point cloud data.

3.1 Storage model

We deviate significantly from the block storage model employed by other DBMS systems for storing point cloud data and opt for a simpler, yet efficient, storing scheme. In our approach a flat table is used for storing the point cloud data, where a different column is used for storing the X, Y, Z coordinates and the 23 properties of each point. As a result, each point is stored as a different tuple in the flat table.

[1]The system, including our extensions, can be downloaded from http://www.monetdb.org

In terms of amount of storage needed, the flat table storage requires more space. However, it is more flexible to exploit compression techniques which are more advantageous for column-stores such as run length encoding.

3.2 Database Loading

In most of the systems, the dominant part of loading stems from the conversion of the LAZ files into CSV format and the subsequent parsing of the CSV records by the database engine. We implemented a binary loader that is tailored to the flat storage model that we chose for storing point cloud data. The loader takes as input a LAS/LAZ file and for each property it generates a new file that is the binary dump of a C-array containing the values of the property for all points. Then, the generated files are appended to each column of the flat table using the bulk loading operator COPY BINARY[2]. Our approach allows us to load and index the full AHN2 dataset that consists of approximately 640 billion points in less than one day, while the point cloud extension of PostgreSQL, based on the results presented in [14], should require almost a week for the same task.

MonetDB uses for the evaluation of geospatial predicates a secondary index, column imprints (c.f. Section 2.1.1), instead of a primary spatial index such as R-tree. Its creation is triggered when it encounters a range query for the first time. Imprints storage comes with a 5-12% storage overhead. For the flat-table storage, MonetDB requires the least total storage mainly due to the columnar organisation and the small amount of storage required by the column imprints index [18].

3.3 Query Model

MonetDB has an SQL interface to the Simple Features Access standard of the Open Geospatial Consortium(OGC) [9] with support for the objects and functions defined in the specification. The spatial query model that is used by MonetDB follows the well established two-step approach of **filtering** and **refinement**.

In the filtering step, the majority of points that do not satisfy the spatial predicate for a given geometry G are identified and disregarded using a fast approximation of the predicate. MonetDB performs the filtering using the column imprints

The refinement step operates on the results of the filtering step that produced a superset of the solution. During this step, the spatial predicate is evaluated against the precise geometry G. The refinement can be very expensive, especially when the geometries are complex. Thus, checking exhaustively each point is not desirable. MonetDB, creates a regular grid over the point geometries selected in the filtering step and assigns each geometry to a grid cell. The spatial relation is then evaluated between each non-empty cell and the geometry G. This allows MonetDB to decide whether a grid cell satisfies or not the spatial relation in a single step. However, for cells that overlap the boundary of the given geometry G, an extra step is needed. All points within such cells have to be checked exhaustively and a decision for each of them individually should be taken. The efficiency of this two steps has been tested in the context of point cloud data using the AHN2 data set. The results are compared with a file-based solution, Rapidlasso LAStools, and PostgreSQL [18].

[2]`https://www.monetdb.org/Documentation/Cookbooks/SQLrecipes/BinaryBulkLoad`

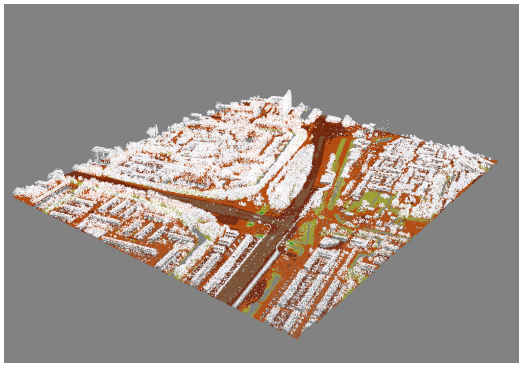

Figure 1: LIDAR point cloud dataset

4. DEMONSTRATION OVERVIEW

During this demo, the users will explore various kinds of geospatial data interacting with the geospatial module of MonetDB, a column-oriented database management system. The demonstration comprises two scenarios. In the first scenario, the audience will be familiarised with the datasets that will be used throughout the demo and will compare the functionality and the performance of a file-based approach and a DBMS based approach. In the second scenario we will stress that a "spatially-enabled" DBMS allows us to run ad-hoc queries for selecting information using both spatial and thematic criteria. Both scenarios will demonstrate the feasibility of using a column-store DBMS as a back-end for geospatial visualisation.

The visualisation tool that will be used is QGIS [6]. QGIS is a cross-platform free and open-source desktop GIS application which supports numerous vector and raster file formats along with connectors to various "spatially-enabled" DBMS. It provides data viewing, editing, and analysis capabilities and allows users to create custom maps that consist of various layers using different coordinate reference systems.

For the purposes of the demonstration, we will use three datasets:

1. *Actueel Hoogtebestand Nederland 2* (AHN2). AHN2 dataset [4] covers the topography of the Netherlands and is freely distributed as open data. The sample density is 6 to 10 points per square meter for the entire country, resulting in 640 billion points organised in 60,185 files. Such files are in LAZ format of total size 640GB. Figure 1 presents a 3D visualisation of the AHN2 dataset.

2. *OpenStreetMap* (OSM). OSM [7] maintains a global editable map that depends on users to provide the information needed for its improvement and evolution. OSM is not restricted to information of small granularity e.g. on national level, but also includes ample information about the road network, the river network, points of interest etc.

3. *Urban Atlas* (UA). UA [8] is a product of the European Environment Agency that provides pan-European information regarding the land use and land cover data for urban zones with more than 100,000 inhabitants. Figure 2 presents a combined 2D visualisation of the UA and OSM datasets.

Below we present the demonstration scenarios in more detail.

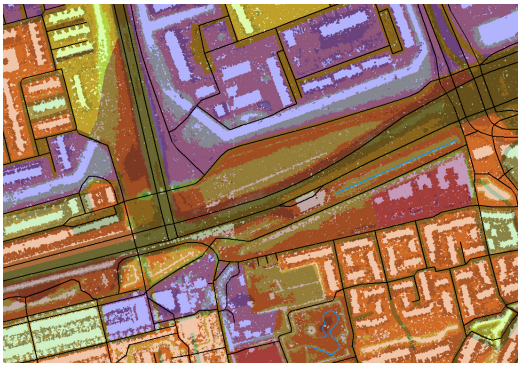

Figure 2: Roads, rivers and land cover data from the OpenStreetMap and Urban Atlas datasets

4.1 First Scenario

The users become acquainted with the datasets and two approaches for handling such data. For this scenario, we will use both a "spatially-enabled" DBMS and a file-based solution for performing a functional and performance comparison between them.

During the functional comparison, the attendees of the demonstration will have the opportunity to compare the expressiveness of the declarative query language SQL, enhanced with geospatial capabilities, to the limited functionality provided by the file-based approach by comparing the queries that can be executed in each of the systems.

During the performance comparison, predefined queries over the three datasets will be executed on both the file-based and the DBMS approach and the time to get the responses will be compared. Since the file-based approach supports only queries over a single data source, the queries that will be executed during this scenario will be of the form "select all LIDAR points within a given region" or "select all roads that intersect a given region", etc..

In order to ease the comparison of the two systems and increase the involvement of the users the results of all queries will be visualised using QGIS [6].

4.2 Second Scenario

In the second scenario, we will stress the fact that a "spatially-enabled" DBMS allows us to run complex queries over multiple datasets. This allows for the discovering of the relations between the various datasets not only through their spatial dimension but through their thematic extent as well.

We will provide the attendees with a set of pre-defined queries like "select all LIDAR points that are near a given area that is characterised as a fast transit road according to the Urban Atlas nomenclature" and "compute the average elevation of the LIDAR points that are near a given area that is characterised as a fast transit road according to the Urban Atlas nomenclature" that will help them discover relations between the various datasets. In addition, the users will have the option to create and execute queries of their own, identifying relations of their interest and obtaining further insights on the datasets.

As before, the results of all queries will be visualized using QGIS. In addition, users will have the option to see the plans of the queries and the execution time spent in each operator. This will give them a more detailed view on the specifities of a column store and the benefits of using it for evaluating spatial queries.

5. CONCLUSIONS

Although column stores have a proven track record in business analytics, their pros- and cons- for GIS applications are not yet well understood. Hence, we have implemented a "spatially-enabled" DBMS which iteratively loads data from different sources and converts it into a common format to enable 3D operations and analyses, and semantic properties management. It bridges the gap between the needs of GIS applications and the available DBMS technologies.

This demo showcases a proof of concept implementation of GIS application in a column-store. The users, through predefined and user defined queries, are able to interact with a point cloud database enriched with semantics from different sources.

6. ACKNOWLEDGMENTS

The work reported here is partly funded by two NLeSC projects, Massive Point Clouds for eScience (project code: 027.012.101) and Big Data Analytics in the Geo-Spatial Domain (project code: 027.013.703), the NWO project COMMIT/, the EU projects Human Brain Project (604102) and LEO (611141).

7. REFERENCES

[1] `http://www.asprs.org/a/society/committees/lidar/lidar\_format.html`.
[2] `http://www.laszip.org`.
[3] `http://www.arcgis.com`.
[4] `http://www.ahn.nl`.
[5] `http://rapidlasso.com`.
[6] `http://www.qgis.org`.
[7] `http://www.openstreetmap.org`.
[8] `http://www.eea.europa.eu/data-and-maps/data/urban-atlas`.
[9] Open Geospatial Consortium. OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option. OpenGIS Implementation Standard, 2010.
[10] P. A. Boncz and M. L. Kersten. MIL primitives for querying a fragmented world. *VLDB J.*, 1999.
[11] P. A. Boncz, S. Manegold, and M. L. Kersten. Database architecture evolution: Mammals flourished long before dinosaurs became extinct. *PVLDB*, 2009.
[12] S. Manegold, P. A. Boncz, and M. L. Kersten. Optimizing main-memory join on modern hardware. *IEEE TKDE*, 2002.
[13] O. Martinez-Rubi et al. A column-store meets the point cloud. *FOSS4G-Europe*, 2014.
[14] S. Ottens. Loading AHN2 into PostgreSQL pointcloud. `http://www.geodan.nl/loading-ahn2-into-postgresql-pointcloud/`, 2014.
[15] H. Sagan. Hilbert's space-filling curve. In *Space-Filling Curves*, pages 9–30. Springer New York, 1994.
[16] L. Sidirourgos and M. L. Kersten. Column imprints: a secondary index structure. In *SIGMOD*, 2013.
[17] M. Stonebraker et al. C-store: A column-oriented DBMS. In *VLDB*, 2005.
[18] P. van Oosterom et al. Massive point cloud data management: design, implementation and execution of a point cloud benchmark. *Computer Graphics*, 2015.
[19] P. van Oosterom and T. Vijlbrief. The spatial location code. *SDH*, 1996.