GI_Forum 2021, Issue 1
Page: 76 - 84
Research Paper
Corresponding Author:
nicole.hein@gia.rwth-aachen.de
DOI: 10.1553/giscience2021_01_s76

Evaluation of a NoSQL Database for Storing Big Geospatial Raster Data

Nicole Hein[1] and Jörg Blankenbach[1]
[1]RWTH Aachen University

Abstract

Database systems capable of efficiently storing geospatial data are widespread. However, recent developments in earth observation systems, remote sensing, mobile mapping, and crowd sourcing lead to large amounts of geospatial mass data that can hardly be handled efficiently with the existing solutions. Especially large geospatial raster data require novel concepts for well-organized data storage.

A concept for storage of large geospatially and temporally referenced image data using the NoSQL graph database system Neo4j as a research subject of the project "RiverView®" is introduced. New strategies and access structures have been developed to ensure the persistence and performant access to image data in Neo4j. These strategies are compared with the up-and download performance of the widespread Rasdaman array database system.

Keywords: geospatial raster database, graph database, big geo data, image database, Neo4j

1 Introduction

Database systems storing spatial-related data (spatial or geodatabase systems) have become standard in the geospatial domain, e.g. as a core component of modern geoinformation systems or distributed spatial data infrastructures. Due to the emergence of novel or further developed geospatial data acquisition methods like mobile mapping systems or multi-sensor earth observation systems, the storage of big geospatial raster data is becoming increasingly important.

This study is part of the research project "RiverView®" and presents an approach for storing of big geospatial raster data within a NoSQL database as well as a benchmark with an existing database system.

2 State of the art and related work

Geodatabase systems are very common in the field of geospatial data management (Bill 2016). Consequently, there are several commercial and free geodatabase systems available like Oracle Spatial and Graph (Oracle Corporation 2021), PostgreSQL/PostGIS (PostgreSQL Global Development Group 2021, PostGIS Project Steering Committee 2021) and MySQL (Oracle Corperations and/or affiliates 2021). Often these storage systems are further developed object-relational databases that have been expanded by specific spatial data types, spatial access structures, and analyses (Yeung & Hall 2007). While in the past, geodatabase systems were mainly applied for storing spatial features as vector data (vector features), more recently, the demand for the efficient storage of spatial raster data has risen. The reason for this development is, in particular, the advent of further developed geospatial data acquisition methods like modern laser-scanning devices, high-resolution digital cameras and novel remote sensing sensors producing large volume raster data sets that require efficient storage (Nebiker 1997). Raster data is represented by (often equidistant) raster cells, which, in the case of raster images, are picture elements and can be stored in implicit structures like matrices or arrays (Nebiker 1997). Brisaboa et al. (2017) described the efficient querying of raster and vector data via k^2- respectively R-Tree data structures. Database systems for storing image data and its classifications have been developed since the late 1970s, e. g. REDI and GRAIN (Chang et al. 1980), (Tamura et al. 1984). In the 1990s, Peter Baumann developed the first prototype of Rasdaman (Rasta Data Manager) to store multidimensional arrays in a database system, especially for geospatial or space sciences (Baumann 1993), (Baumann et al. 1997). Rasdaman is a middleware working with PostgreSQL and SQLite on a storage basis (Baumann 2018). Since the term Big Data arose (Chalmers et al. 2013) with its different types and particular challenges (Lansley et al. 2019) arises, NoSQL database systems are becoming increasingly popular. In (DeZyre 2019) several reasons are identfied for using NoSQL database systems in terms of Big Data because relational database systems are not suitable for the complexity and heterogeneity of upcoming data. Additionally, NoSQL database systems are easily expandable. Since Big Data is complex and contains highly interconnected information, it is represented well as a graph (Miller 2013).

3 Background

The research project RiverView® (FiW 2020) aims at developing a novel approach for the holistic monitoring of medium and small watercourses. The core component of RiverView® is an unmanned surface vehicle (RiverBoat, Fig. 1) equipped with multiple sensors, which allows for autonomous digital water data acquisition with high spatial and temporal resolution. In addition to chemical-physical sensors, an above-water mapping system is installed , containing an omnidirectional multi-camera system consisting of 6 individual cameras, with which georeferenced images (5 MP each) of the water environment can be recorded continuously at high temporal frequency (max. 10 Hz).

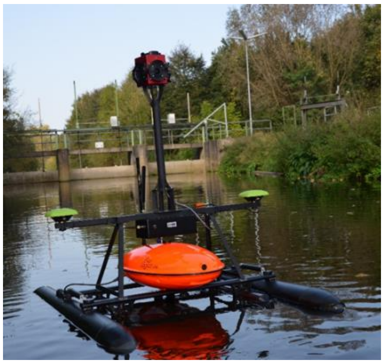
Figure 1: RiverBoat

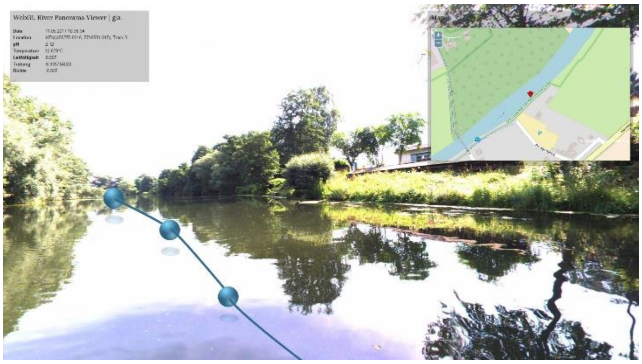
Figure 2: Water body information system

For managing all collected data, a GIS-based water body information system (Fig. 2) was developed. Therefore, efficient and powerful storage capacities due to the heterogeneous and large volume datasets are required. Whilst the scalar and vectorial data (e.g. O_2 level, water temperature) can be inputted directly in a relational geodatabase, image data storage in particular is a major challenge because it has to fulfil the following characteristics:

1) (Near) real-time data export
2) Scalability / Big Data ability
3) Handling heterogenous data

Pre-existing solutions (e.g. PostgreSQL/PostGIS, Oracle) have been tested for geospatial image storage (Hein & Blankenbach 2017). However, after concluding evaluations, several problems (e. g. no real-time ability, no Big Data ability) were identified.

Hence, a novel concept was developed for storing geospatial raster data based on the NoSQL database system Neo4j. Neo4j (Neo Technology 2018) is a graph database that includes the topological components "node", "relationship", "property" and "label" as well as data indexing features to find nodes as basic information item faster in the graph. For geospatial data handling Neo4j provides a spatial library including e. g. spatial search trees (R-Trees) to accelerate read operations on spatial data (Taverner 2019).

4 Raster Data Storage and Indexing

A geospatial raster image is a matrix of picture elements (pixels) consisting of colour and possibly transparency information. In practice, two approaches are commonly pursued for the database-driven storage of geospatial raster images. Either the raster images are stored directly in the database (e. g. by using Binary Large Object (BLOB)) or using an image file format on the hard disk. In both cases, the metadata (e.g. spatial reference, image dimensions and resolution) are stored in the database that enables the deployment of useful access structures to the data.

Database indices are utilised as efficient access structures that have to be optimised, in this case for raster data in different resolutions. For the latter image, pyramids are used to provide the original image in different resolutions (Fig. 3, left).

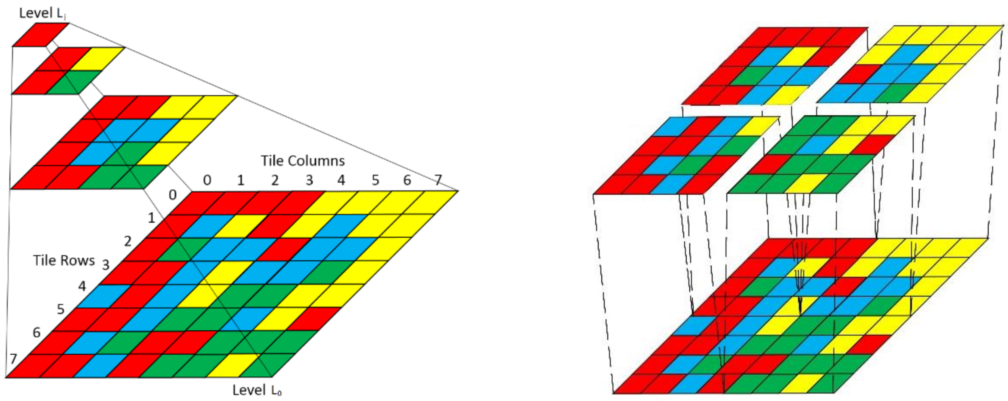

Figure 3: Image pyramid and adapted tiling (Source: Esri 2021)

Another crucial database access structure for raster data is tessellation (Fig. 3, right). For real-time applications, the ability to load only parts of images is necessary, hence, images must be split into delimited parts. Therefore, rectangular blocks are normally applied, which is why this process is also commonly known as tiling. However, in general, different tiling strategies can be used, e.g.:

1) Aligned Tiling: The Aligned Tiling Strategy (ATS) divides the image data into rectangular tiles with equal height and width (Fig. 4, left top).
2) Random Tiling: The Random Tiling Strategy (RTS) calculates for each rectangular tile an individual height and width randomly (Fig. 4, left bottom).
3) Region-of-Interest Tiling Strategy (RoITS): This tiling strategy was explicitly developed for the RiverView application because large parts of the images contain water or sky, which are less relevant to users. Thus, the general idea was to tile only the interesting areas in the image in a more granular way. The RoITS, therefore, identifies points of interest (POI), e.g. by calculating the image feature points using the SIFT (Lowe 2004) algorithm (Fig. 4, right). The more feature points found in an area, the more granular the area of the image is tiled: If a certain amount of points are found in a rectangle, a tile is defined.

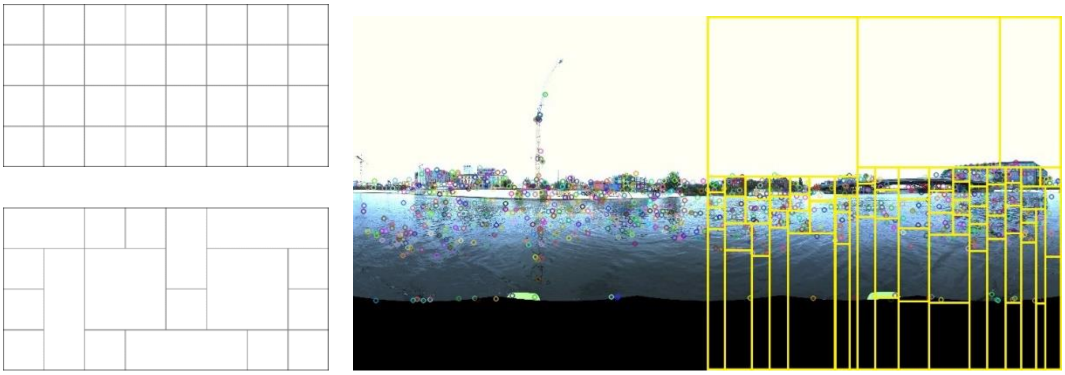

Figure 4: ATS (left top), RTS (left bottom) and RoITS (right)

Both, image pyramids and tiling, is then used to create a spatial index, speeding up spatial queries (e.g. query boxes) on the data. In geodatabases, search trees usually represented by graphs are applied and are stored separated from the data itself. A very widespread spatial search tree is an R-Tree (Fig. 5) (Guttman 1984).

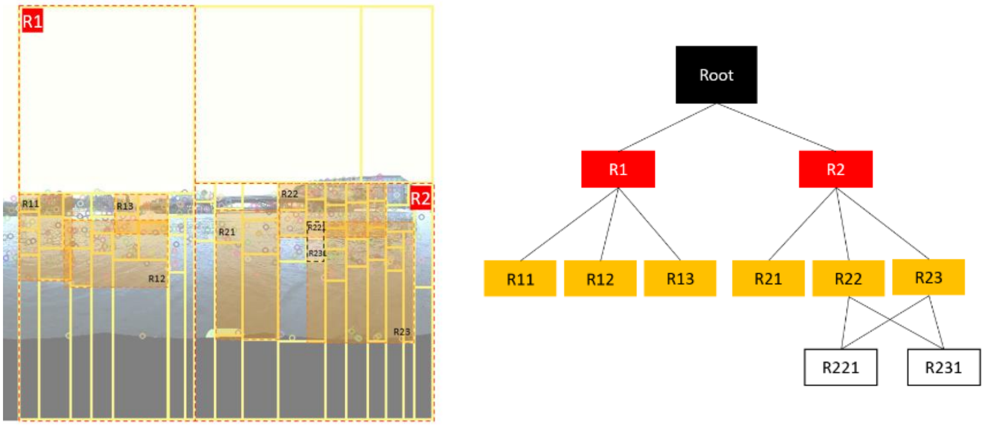

Figure 5: R-Tree on image (right) and R-Tree as graph (left)

5 Implementation

For the implementation of the raster data storage in Neo4j, it had to be decided whether only the metadata and the access structures or also the image data itself should be stored in the database (see section 4). Out of preliminary tests, the storage of binary (raster) data directly in the database is not efficient, which is also confirmed by (Armbruster 2016). Hence, the raster data is stored on hard disk while Neo4j holds the metadata.

Thus, for each image, the following steps were conducted:

1) An image pyramid (Gaussian pyramid) is created.
2) Each level of the pyramid is tiled using the respective variants ATS, RTS and RoITS.
3) A search tree (R-Tree) is created for each pyramid level.
4) The tilesets are stored on a hard disk while the metadata, a link of each tile, and the R-Tree is stored into Neo4j.

For the implementation of these four consecutive steps, OpenCV and NumPy were used. Uploading images in Neo4j is done via Python with the extension Neo4j Spatial. In the database, only the extends of the image tiles are stored as polygons and the spatial reference as point positions.

For the subsequent benchmark of our new raster data management concept, the images were imported additionally into Rasdaman, a powerful image database storing multidimensional array data.

6 Evaluation

Whilst data access structures lead to a performance gain at read access, they cause a decrease in performance for write access. Hence, for benchmarking an evaluation between the two databases, Rasdaman and Neo4j, regarding up- and download performance was conducted. Both databases (Neo4j v3.4.9; Rasdaman v9.6) were installed on the same computer with Linux Debian 8 (6GB RAM).

Fig. 6 shows the upload results of the different tiling strategies with varying tiling sizes (480 x 480, 500 x 500, 1000 x 1000) on the abscissa axis and the time in seconds on the ordinate axis. For RoITS at 50 POI, the tiles are 481 x 633 pixels on average. At 100, POI the tiles have an average size of 554 x 805 pixels. RoITS needs the longest time for uploading data – no matter which size of POI is considered. ATS and Rasdaman show a fast upload time.

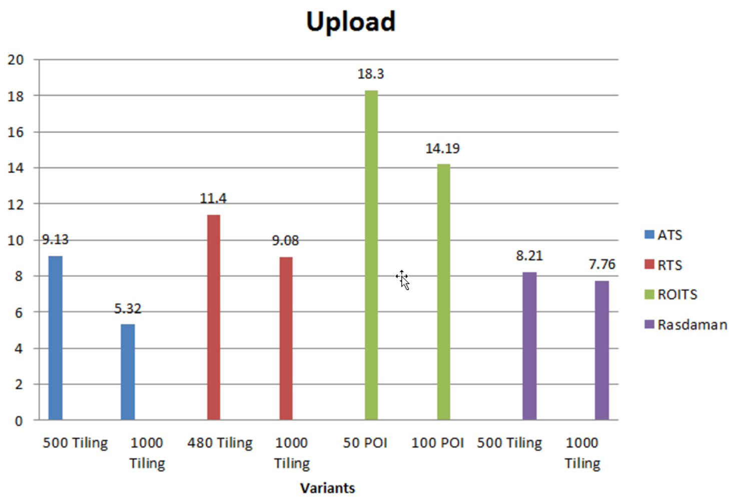

Figure 6: Upload benchmark with comparison to Rasdaman

The download test is triggered by specifying spatial coordinates of a section of the image which is required. Hence, a bounding box (range) query with the geometric function "intersects" is executed in the database. Furthermore, the corresponding tiles within the bounding box are loaded from the database. Fig. 7 depicts the average download time for the resolutions of the pyramid levels 0 to 3 when the entire image (8000 x 4000 pixels) is loaded. It is noteable that Rasdaman (both tiling sizes) and ATS (500 x 500 pixels) take the longest time, taking approximately 3 seconds for the download. All strategies vary between 1.15 and 1.62 seconds. Fig. 8 shows the download times in the resolution levels 0 to 3 of the pyramid level with a bounding box of (x_1, y_1, x_2, y_2) = (3526, 512, 7654, 3709). Considering the number of nodes, it can be concluded that the higher the number of generated nodes there is, the more time the download takes (see Table 1, Fig. 7 and Fig. 8).

A similar result to the upload emerges: Rasdaman (both strategies) and ATS with 500 x 500 tiling require the most time. All other strategies require between 0.66 and 0.92 seconds.

Table 1: Number of nodes for strategies

Strategy	Size	Number of nodes
ATS	500 x 500 pixels	13,086
ATS	1000 x 1000 pixels	3,349
RTS	480 x 480 pixels	5,317
RTS	1000 x 1000 pixels	2,520
RoITS	50 POI	6,601
RoITS	100 POI	4,573

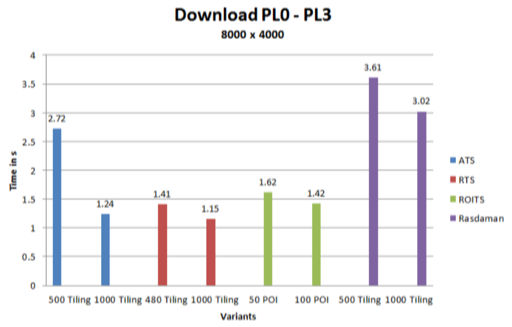

Figure 7: Download benchmark, 8000 x 4000

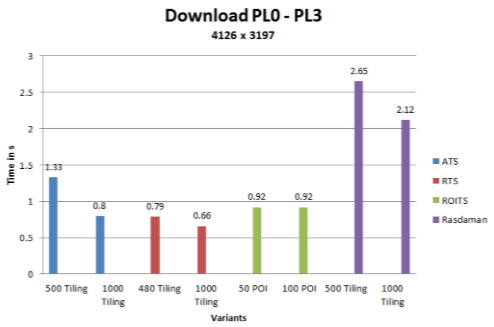

Figure 8: Download benchmark, 4126 x 3197

7 Summary and Conclusion

The collection of large amounts of geospatial image data in the project RiverView® requires efficient data storage. Since existing solutions do not offer an optimal solution, a raster data storage concept based on the NoSQL database system Neo4j was developed and implemented. A crucial aspect is the implementation of data access structures such as image pyramids and tiling. Hence, different tiling strategies were evaluated and benchmarked for up- and download. Based on these benchmarks it evident that the RTS and RoITS tiling strategies perform best in download. Generally, the strategy implemented depends on the application type. In summary, efficient geospatial rasta data management with Neo4j is possible based on the developed strategies and can even be used for real-time applications. It is also conceivable to extend the approach to remote sensing and satellite data sets for write-once-read-many use-cases. In future work, the approach will be further developed to point clouds and also additional evaluations considering other solutions (e.g. Open Data Cube) are planned.

Reference

Armbruster, S. (2016). Welcome to the Dark Side: Neo4j Worst Practices (& How to Avoid Them). https://neo4j.com/blog/dark-side-neo4j-worst-practices/

Baumann, P. (1993). Language support for Raster Image Manipulation in Databases. *Visualization in Science & Technology.*

Baumann, P., Furtado, P., Ritsch, R. & Widmann, N. (1997). Anfrageformulierung und Ablage dimensionsbehafteter Daten in Rasdaman. https://doi.org/10.1007/978-3-642-60730-1_17

Baumann, P. (2018). Rasdaman – Raster data manager. https://rasdaman.com

Bill, R. (2016): Grundlagen der Geo-Informationssysteme. 6. Auflage, Wichmann-/VDE-Verlag, Berlin

Brisaboa, N., de Bernardo, G., Gutiérrez, G., Luaces, M. R. & Paramá, J. R. (2017), Efficiently Querying Vector and Raster Data. *The Computer Journal*, Volume 60, Issue 9, September 2017, Pages 1395–1413, https://doi.org/10.1093/comjnl/bxx011

Chalmers, S. & Bothorel, P.-C. C. (2013). Big Data – State of the Art. *Report, Télécom Bretagne.*

Chang, N. S. & Fu, K. S. (1980). A relational database system for images. *Pictorial Information Systems*: Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-09757-0_11

DeZyre (2019). NoSQL vs SQL – 4 Reasons why NoSQL is better for Big Data applications. https://tinyurl.com/y2yrmkqn

Esri (2021). Tiled Elevation Service. https://developers.arcgis.com/documentation/tiled-elevation-service/

FiW Aachen (2020). RiverView®. https://www.river-view.de

Guttman, A. (1984). R-Trees: A Dynamic Index Structure for Spatial Searching. *Proceedings of the 1984 ACM SIGMOD international conference on Management of data – SIGMOD '84.* https://doi.org/10.1145&602259.602266

Hein, N. & Blankenbach, J. (2017). Vergleich von PostGIS und Rasdaman als Geodatenbank für großvolumige Bilddatenbestände eines mobilen Mappingsystems. In J. Strobel, G. Griesebner, & T. Blaschke (Eds.), *AGIT 3-2017 – Journal für Angewandte Geoinformatik*: Wichmann Verglag. https://doi.org/10.14627/537633001

Lansley, G., de Smith, M., Goodchild, M. & Longley, P. (2019). Big Data and Geospatial Analysis. *Computers and Society.* https://arxiv.org/abs/1902.06672

Lowe, D. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision.*

Miller, J. J. (2013). Graph Database Applications and Concepts with Neo4j.

Nebiker, S. (1997): Spatial raster data management for geo-information systems – a database perspective. Doctoral thesis, ETH Zurich. https://doi.org/10.3929/ethz-a-001847022

Neo Technology (2018). Neo4j. https://neo4j.com/

NumPy Developers (2018). NumPy. https://www.numpy.org/

Oracle Corperations (2021). Spatial and Graph features in Oracle Database. https://www.oracle.com/database/technologies/spatialandgraph.html

Oracle Corperations and/or affiliates (2021). MySQL. https://www.mysql.com

OpenCV Dev Team (2019). OpenCV. https://www.opencv.org/

Pathak Siddharth, D., Thakur Siddharth, J., Shitole Siddharth, J., Gaikwad, D. P. & Satam Tejas, S. (2014). Implementation of R-Trees for Spatial Image Processing and Cloud Detection., *International Journal of Engineering Research & Technology (IJERT).* ISSN: 2278-0181

PostGIS Project Steering Committee (2021). PostGIS. https://postgis.net/

Tamura, H. & Yokoya, N. (1984). Image database systems: A asurvey. *Pattern Recognition 17-1.* https://doi.org/10.1016/0031-3203(84)90033-5

Taverner, C. (2019). Neo4j Contrib Spatial. https://neo4j-contrib.github.io/spatial/

The PostgreSQL Global Development Group (2021). PostgreSQL. https://postgresql.org/

Yeung A.K.W., Hall G.B. (2007): Spatial Database Systems. The GeoJournal Library, vol 87. Springer, Dordrecht. https://doi.org/10.1007/1-4020-5392-4_4