International Journal of Geo-Information

MDPI

Review

State-of-the-Art Geospatial Information Processing in NoSQL Databases

Dongming Guo * and Erling Onstein

Department of Manufacturing and Civil Engineering, Norwegian University of Science and Technology, 2802 Gjovik, Norway; erling.onstein@ntnu.no

* Correspondence: dongming.guo@ntnu.no; Tel.: +47-92559641

Received: 29 January 2020; Accepted: 15 May 2020; Published: 19 May 2020

Abstract: Geospatial information has been indispensable for many application fields, including traffic planning, urban planning, and energy management. Geospatial data are mainly stored in relational databases that have been developed over several decades, and most geographic information applications are desktop applications. With the arrival of big data, geospatial information applications are also being modified into, e.g., mobile platforms and Geospatial Web Services, which require changeable data schemas, faster query response times, and more flexible scalability than traditional spatial relational databases currently have. To respond to these new requirements, NoSQL (Not only SQL) databases are now being adopted for geospatial data storage, management, and queries. This paper reviews state-of-the-art geospatial data processing in the 10 most popular NoSQL databases. We summarize the supported geometry objects, main geometry functions, spatial indexes, query languages, and data formats of these 10 NoSQL databases. Moreover, the pros and cons of these NoSQL databases are analyzed in terms of geospatial data processing. A literature review and analysis showed that current document databases may be more suitable for massive geospatial data processing than are other NoSQL databases due to their comprehensive support for geometry objects and data formats and their performance, geospatial functions, index methods, and academic development. However, depending on the application scenarios, graph databases, key-value, and wide column databases have their own advantages.

Keywords: geospatial data; NoSQL databases; spatial index; geospatial functions; data models

1. Introduction

The amount of personal location data is forecast to increase by 20% every year, and location-aware information occupies a large proportion of the data generated every day: 2.5 quintillion bytes [1,2]. The advent of the geospatial big data era requires new applications and creates new challenges [3,4]. How to store, manage, and query geospatial data effectively have become the focus of research and are problems that must be solved [3,5–7]. At present, the main geospatial databases are divided into two types: relational databases and NoSQL databases. Relational databases are the most widely used and the most mature database systems, and they have been applied in various industries for decades. To enrich geospatial functions and geospatial processing capability, some modern relational databases have made some changes and updates. Examples of relational databases for geographic information include PostGIS [8], WebGIS [9], Oracle 19c [10], the Microsoft Azure SQL Database [11], and the SQL Server [12]. These relational databases can define geospatial objects, support the main spatial data types (for geometry), and adopt different indexes for fast spatial queries (Binary Tree in SQL Server, Binary Tree, R-Trees, and Generalized Search Tree in PostGIS). Additionally, most applications using spatial relational databases are desktop systems (such as ArcGIS) or have map server software (such as

GeoServer). Traditional relational databases adopt fixed structure/data schemas, and their scalability is limited.

NoSQL databases are general distributed database systems, which may not require structured data, are typically designed for scaling horizontally, and may be open source [5,13]. For horizontal scalability, NoSQL databases do not provide the standard ACID properties (atomicity, consistency, isolation, and durability) that are provided by relational databases. However, NoSQL databases exhibit the ability to store, manage, and index arbitrarily big datasets while supporting a large number of concurrent user requests [14]. Currently, NoSQL databases are now being widely used in various application fields [4,15–17].

With the development of mobile communications, the IoT, and high-speed network access technologies, the need for geographic information applications for mobile services and web services has become increasingly strong [2–4]. New geospatial applications require more flexible data schema, a relatively fast query response time, and more elastic scalability than traditional spatial relational databases currently have. For example, when the streaming requests from clients to servers suddenly increase, it might cause significant response delays and service unavailability. To solve this scalability problem, a scalable framework was proposed based on MongoDB to implement elastic deployment for geospatial information sharing with the client users growing in number [14]. In this framework, MongoDB is chosen because it is a distributed database and supports a flexible storage schema suitable for massive map tile storage [14].

Several studies have found that relational database management systems (RDBMS) have some disadvantages in terms of big data storage and queries in some specific areas, such as in high concurrent or large-scale data access environments in geospatial applications [4,5]. In one qualitative comparison of experiments, it was found that document databases have faster response times and line intersection queries than SQL databases when the number of records in the databases is large [5,18]. Other studies have indicated that NoSQL databases have more advantages in geospatial data processing than relational databases [6]. In testing of the most used spatial query functions in different databases, NoSQL databases performed better than did relational databases, especially mobile-GIS and Web-GIS [18,19]. Currently, most NoSQL databases are viewed as not being well designed for geospatial data [6,20]. One of the most obvious deficiencies of NoSQL databases in terms of geospatial data is that NoSQL databases only have basic spatial functions, far fewer than relational databases have [6]. Fortunately, research on NoSQL databases is a burgeoning field, attracting more and more attention from enterprises and academics, and improvements and innovations have rapidly emerged. Moreover, some NoSQL databases have some spatial functions and spatial indexes [18,21]. In recent years, some academic articles have summarized and analyzed the applications of SQL and NoSQL databases in geographic data fields, including one study of geospatial big data [3], a summary of the best practices for publishing spatial data on the web [22], an experimental comparison between two geospatial data platforms [19,23], comparisons between relational databases and NoSQL databases in geospatial applications [5,18], and a study on geospatial semantic data management [24]. However, there have been few comprehensive analyses of state-of-the-art geographic data processing in popular NoSQL databases.

To elaborate on state-of-the-art geographic data processing in popular NoSQL databases, in this paper, we first introduce the geospatial data characteristics and related concepts and then review state-of-the-art geospatial data processing in the 10 most popular NoSQL databases. Moreover, we analyze the pros and cons of these NoSQL databases for geospatial data processing. The paper structure is as follows. Section 2 introduces the geospatial data characteristics and related concepts. Section 3 introduces the research methodology adopted for this paper. State-of-the-art geospatial processing in NoSQL databases is presented in Section 4. Section 5 compares the performances of the different NoSQL databases in terms of geospatial data storage and queries. Section 6 includes a brief conclusion.

2. Geospatial Data Characteristics and Related Concepts

Before discussing geospatial data processing, here we introduce the basic geospatial concepts and characteristics of geospatial science.

Generally, there are two main ways to represent geospatial data: raster and vector data.

- Raster data are made up of a matrix of cells (grain or pixels), in which a cell has an associated value representing information, such as a brightness value or temperature, and are arranged into rows and columns (or a grid).
- Vector data consist of individual points that are stored as pairs of (x, y) in 2D cases or (x, y, z) in 3D cases. The points are connected through certain orders/rules to create lines, polygons, surfaces, and solids. In this paper, most of the discussions refer to vector data.

Features and geometries are the two main foundational concepts. A feature can be any object with a given spatial location, such as an airport or a mountain.

- According to ISO 19109:2015, a feature is defined as the "abstraction of real-world phenomena". Features may have attributes, e.g., spatial attributes giving the location/extent of the feature, thematic attributes giving descriptive characteristics of the feature, and also other kinds of attributes, such as metadata/quality.
- The geometry is any geometric shape that can represent a feature's spatial attribute, such as a point (0D), line (1D), polygon/surface (2D), or solid/volume (3D). The geometries can be embedded in 1D space, 2D space, or 3D space. The dimension of the geometry must be smaller than or equal to the dimension of the embedded space. For simple cases such as visualization using traditional 2D maps (2D space), points, lines, and polygons might be sufficient for user needs. For more complex cases requiring 3D space, surfaces and volumes/solids are also required.

In NoSQL databases, the main geometry types include Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon, and GeometryCollection.

In relational databases, based on the international standard ISO/IEC 13249: 2016, SQL Multimedia Application Packages provide 27 geometry types, of which 24 geometry types are instantiable and have constructor functions. The geometry types and methods in relational databases are obviously more abundant than in NoSQL databases.

To handle the features' spatial locations and relationships, coordinate reference systems are needed.

- A coordinate reference system (CRS), or a spatial reference system (SRS), is a coordinate-based system for locating geographical entities and establishing their relationships. Popular coordinate reference systems include the geocentric coordinate system, geographic coordinate system (WGS84 datum), Universal Transverse Mercator (UTM), and Cartesian coordinate system.
- In coordinate reference systems, Well-known Text (WKT) is a text markup language that represents coordinate reference systems and conversions between different coordinate reference systems, as defined by the Open Geospatial Consortium (OGC).
- The EPSG Geodetic Parameter Dataset (also called the EPSG registry), which was developed by the European Petroleum Survey Group (EPSG) in 1985, is a public collection of the definitions of coordinate reference systems and coordinate transformations. The EPSG code is widely used in geographic information systems and GIS libraries.

Additionally, topological relationships are critical to geospatial processing and data queries. Some issues with instance relationships can be solved through features' topological relationships. In addition to the traditional graph topology, there are three popular topological relation principles: implicit topology as in simple features [25] and point-set topology as in Egenhofer nine-intersection relations [26] and in RCC8 relations [27]. Battle et al. summarized the equivalence between the three spatial relations, as shown in Table 1 [28].

Table 1. The equivalency relationships between simple features, Egenhofer, and RCC8.

Simple Features	Egenhofer	RCC8
equals	equal	EQ
disjoint	disjoint	DC
intersects	¬disjoint	¬DC
touches	meet	EC
within	inside + coveredBy	NTPP + TPP
contains	contains + covers	NTPPi + TPPi
overlaps	overlap	PO

Note: "¬" means "not".

The special characteristics of geospatial data (especially multi-dimensionality and the large size of datasets) make processing them different than processing other data. These data require different platforms, flexible scalability, and the ease of modification, update, and query, such as large-scale spatiotemporal query scenarios [29] and huge access requests based on mobile platforms [14].

3. The Research Methodology

Many NoSQL databases and related products are developed and updated continually currently. To compare and discuss the geospatial development of NoSQL databases effectively, the 10 most popular NoSQL databases (as ranked by DB-Engines [30]) were chosen to analyze their characteristics and performances in terms of geospatial data processing. These NoSQL databases fall into six types of database models: document databases, graph databases, wide column databases, key-value databases, multi-model databases, and search engine, as listed in Table 2. We used the name of each of the 10 databases and "geospatial OR spatial" as search keywords to search for articles in general academic databases, including the Web of Science Core Collection, Google Scholar Citations, and the Scopus database. After filtering some duplicated and unrelated articles, we obtained our final set of articles; how many were found in each database is shown within round brackets in Table 2. Through an analysis of NoSQL databases and related research, we explored the state-of-the-art of geospatial information processing in NoSQL databases. Few NoSQL databases support 3D data scenes, so if there is no special annotation, the geometry functions and indexes indicate 2D data scenes.

Table 2. Mainstream NoSQL databases and the number of related articles found within them.

Database Model	NoSQL Databases
Document	MongoDB (21), Couchbase (1)
Graph	Neo4j (6)
Wide column	Cassandra (5), HBase (24)
Key-value	Redis (2)
Multi-model	Amazon DynamoDB (0)
Search engine	Elasticsearch (3), Splunk (0), Solr (3)

4. State-of-the-Art Geospatial Processing in NoSQL Databases

In this section, basic information about these databases is shown in Table 3. After that, some succinct information about the geospatial characteristics in these databases is introduced. Additionally, related research from the different databases is also summarized.

Table 3. Basic information about the 10 NoSQL databases. DBMS, relational database management system.

Database Name	Primary Database Model	Secondary Database Models	Website	Developer	Server Operating Systems	APIs and Other
MongoDB	Document		www.mongodb.com	MongoDB, Inc	OS X, Windows	Proprietary protocol using JSON
Couchbase	Document	Key-value	http://www.couchbase.com	Couchbase, Inc.	OS X, Windows	Native language bindings for CRUD, query, search and analytics APIs
Neo4j	Graph		neo4j.com	Neo4j, Inc.	Linux, Unix, Windows	Bolt protocol, Cypher query language, Java API, Neo4j-OGM, RESTful HTTP API, Spring Data Neo4j
Cassandra	Wide column		cassandra.apache.org	Apache Software Foundation	BSD, Linux,	Proprietary protocol
HBase	Wide column		hbase.apache.org	Apache Software Foundation	Linux,	Java API, RESTful HTTP API, Thrift
Redis	Key-value	Document, graph DBMS, search engine	redis.io	Salvatore Sanfilippo	Linux, OS X, Solaris, Windows	Proprietary protocol
Amazon DynamoDB	Document, Key-value		aws.amazon.com/dynamodb/	Amazon	Hosted	RESTful HTTP API
Elasticsearch	Search engine	Document	https://www.elastic.co/elasticsearch/	Elastic	Linux, OS X, Solaris, Windows	Java API, RESTful HTTP/JSON API
Splunk	Search engine		www.splunk.com	Splunk Inc.	BSD, Linux, OS X, Windows	HTTP REST
Solr	Search engine		https://lucene.apache.org/solr/	Apache Software Foundation	All OS with a Java VM	Java API, RESTful HTTP/JSON API

4.1. MongoDB

MongoDB is a document database that stores data in scalable, flexible, JSON-like documents, with different data fields and a changeable data structure. MongoDB does not support a declarative query language: queries in MongoDB are built and issued by proprietary API or drivers. MongoDB supports storing and querying geospatial data. To describe GeoJSON data, MongoDB uses an embedded document with a GeoJSON object type and then the object's coordinates, listing the longitude first and then the latitude:

<field>: {type: <GeoJSON type>, coordinates: <coordinates> }
For example, to specify a GeoJSON Point:
location: {
type: "Point",
coordinates: [-43.342412, 54.678421]
}

MongoDB supports multiple geometric objects, as listed in Table 4. Additionally, MongoDB uses the WGS84 reference system for geospatial queries of GeoJSON objects. Valid longitude values are between −180 and 180 (inclusive), and valid latitude values are between −90 and 90 (inclusive).

MongoDB stores object location data as legacy coordinate pairs and supports spherical surface calculations via a 2dsphere index, of which there are two data representations: an array (MongoDB preferred) and an embedded document in legacy coordinate pairs:

An array: *<field>: [<x>, <y>]*
An embedded document: *<field>: {<field1>: <x>, <field2>: <y> }.*

Table 4. A Comparison of NoSQL databases in terms of their geospatial features. WKT, Well-known Text; KMZ, Keyhole Markup Zipped; KML, Keyhole Markup Language.

Database	Supported Geometry Objects	Main Supported Geometry Functions	Supported Spatial Indexes	Declarative Query Language	Data Format
MongoDB	Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeometryCollection	$geoIntersects, $geoWithin, $near, $nearSphere	2dsphere index, 2d index based on geohash	not supported	GeoJSON objects, Legacy Coordinate Pairs
Couchbase	Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeometryCollection	BBox	R-Tree, designed by users	N1QL	GeoJSON
Amazon DynamoDB	Point	GeoPoint, putPoint deletePoint, updatePoint queryRectangle, queryRadius	geohash	not supported	GeoJSON
Cassandra	Point, LineString, Polygon	intersects, contains, is_within (provided by a plug-in)	Lucene index (provided by a plug-in)	Cassandra Query Language (CQL)	WKT
HBase	No	No	No	not supported	No
Neo4j	Only Point in Neo4j; Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeometryCollection in Neo4j Spatial Library	distance(), point()-WGS 84 2D, point()-WGS 84 3D, point()-Cartesian 2D, point()-Cartesian 3D	B+Tree, R-Tree (default)	Cypher	Graph Model, RDF
Elasticsearch	Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeometryCollection	geo_shape, geo_bounding_box, geo_distance, and geo_polygon	GeohashPrefixTree and QuadPrefixTree	Domain Specific Language (DSL)	GeoJSON and WKT
Redis	Point	geoadd, geodist, geohash, geopos, georadius and georadiusbymember	geohash	not supported	GeoJSON
Solr	points, circles, envelopes, line strings, polygons, and “multi” variants of these	convexHull, enclosingDisk,	Prefix tree, geohash, and quadtree	“Lucene” query parser (default)	WKT or GeoJSON
Splunk	Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeometryCollection	Geom inputlookup	Summary indexing	Splunk Process Language (SPL)	KMZ or KML

MongoDB provides two geospatial index types: geohash for 2dsphere and 2d. Within 2dsphere indexes, relevant queries are implemented through a calculation of geometries on an Earth-like sphere. Within 2d indexes, queries are implemented through a calculation of geometries on a two-dimensional plane. For spherical queries, 2dsphere indexes should be used, because the use of 2d indexes for spherical queries may lead to incorrect results. Four topological query operations are provided in MongoDB for geospatial data: $geoIntersects, $geoWithin, $near, and $nearSphere.

In a quantitative comparison of geospatial big data processing between the PostGIS and MongoDB databases, MongoDB had some advantages with its “within” and “intersection” queries [18,31] and in terms of its response time for loading big geospatial data [32]. Meanwhile, in a comparison between ArcGIS and MongoDB, the spatial retrieval performance of MongoDB was better than that of ArcGIS, and this advantage was more obvious with an increase in the point set [33].

In related research on MongoDB, attention has focused on geospatial data management and storage [34–43] and on index development methods [29,44–46]. Using MongoDB, several platforms and frameworks have been designed to solve diverse application demands. Although NoSQL databases can

store JSON objects, a standard query language is still missing, so individuals who are not programmers have a hard time managing, analyzing, and correlating geospatial data. To solve these problems, a framework and a query language were designed to manipulate JSON objects and provide spatial and non-spatial operations across heterogeneous datasets [34]. To provide a multi-user collaborative work environment, a prototype system was developed based on open-source web technologies, in which geospatial data were processed according to the OGC standard and modified as a GeoJSON format to be stored in MongoDB [43]. When multiuser requests to servers increase to a certain extent, response times and service might be subpar or unavailable. For this scale problem, a scalable Web Map Tile Services (WMTS) framework was designed with a high-performance cluster to implement elastic deployment as user requests grow in number [14]. Aside from the problems of multiple user requests, the management of very large geospatial datasets is also an urgent problem. A software, called GeoRocket, has been created to manage very large geospatial datasets in the cloud [36]. GeoRocket splits large datasets into chunks and processes chunks individually. GeoRocket has adopted Elasticsearch to index and query large datasets and uses MongoDB for data storage [36].

Although MongoDB provides index methods, research is still being done to find faster indexes. Xiang et al. proposed a method of implementing an R-Tree index, which combines spatial range query and nearest neighbor query in MongoDB [44]. Using a tabular document structure, they flattened the R-Tree index into MongoDB collections, and the experiment showed that the new method performed better than the 2dsphere index (MongoDB's built-in spatial index) [44]. The other method used for R-Tree in MongoDB was proposed by Li et al., while a geohash-based spatial index has been applied in location-based queries for a medical monitoring system, which combined nested minimum boundary rectangles (MBRs), an R-Tree as a global tree for real-time locations, and a geohash-based B-Tree as a local tree for historical data [45]. Additionally, some researchers have also made contributions using MongoDB to the spatiotemporal index design of massive trajectory data [29,47].

MongoDB provides a wide and flexible platform for different geospatial applications. Moreover, MongoDB adopts some tactics to improve performance and availability, such as asynchronous replica updates and load balance across replicas, but these tactics can affect the consistency of one or multiple objects. This also happens with other NoSQL databases, such as HBase, Cassandra, and Neo4j.

4.2. Couchbase

The Couchbase Server is an open-source, distributed, document-oriented database with fast key-value storage and a powerful query engine for executing an SQL-like query language (N1QL) [48]. The Couchbase Server is designed for some specific environments to provide low-latency data management services, such as a large-scale interactive web, a mobile terminal, or IoT applications. The Couchbase Server supports some geometry primitives, as does MongoDB. For geospatial queries, it has two location representation models: radius-based and box-based. In the radius-based location representation model, location data are shown as locations with longitude–latitude coordinate pairs and the distance in miles. This distance is the length of the radius, and the location of an object is in the center of the circle. If the query location is within the circle, documents are returned. In the box-based location representation model, two longitude–latitude coordinate pairs are required, which are located at the top-left and bottom-right corners of a box. If the query location is within the box, JSON documents are returned, and they contain the location within the box. The Couchbase Server provides R-Tree indexes for location-aware applications. Additionally, spatial indexes can also be defined by users before a geospatial query. Depending on which of the two location representation models are used (a location or a bounding box), the spatial indexes are different. Couchbase only provides queries based on location coordinate data, which can limit its applications.

There was one article about the Couchbase database in the search results, which is mentioned in Section 3. In it, an information-centric network was adopted to federate MongoDB and Couchbase databases [35]. The functional architecture of the designed federated database included a federation front-end for effective connection between a query processor and users; the query processor for

interacting with local and remote DBSes; and a DBMS adapter for translating the federated query into the local query language in a local DBMS [35].

4.3. Neo4j

Neo4j is the most popular graph database and uses Cypher as its query language, but it supports only one type of spatial geometry, Point, in the latest Version 3.5. Each point in Neo4j can have 2D or 3D presentation and can be specified as a geographic coordinate reference system or a Cartesian coordinate reference system. Because Neo4j has only one spatial geometry type, the database provides spatial functions related to the point, such as distance(), point()-WGS 84 2D, point()-WGS 84 3D, point()-Cartesian 2D, and point()-Cartesian 3D. An example from Cypher's manual [49] is:

```
WITH point ({latitude:toFloat('13.43'), longitude:toFloat('56.21')}) AS p1,
point ({latitude:toFloat('13.10'), longitude:toFloat('56.41')}) AS p2
RETURN toInteger(distance(p1, p2)/1000) AS km
```

There is a utility library called Neo4j Spatial that facilitates the spatial manipulation of data. Neo4j Spatial supports seven common geometry types: Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, and GeometryCollection and topology operations. Additionally, Neo4j Spatial adopts an R-Tree index for spatial queries and provides multiple spatial procedures [50].

In related research, Sarwat et al. implemented a reachability query with a spatial range predicated on the Neo4j graph database, trying to find whether an input point can reach any spatial point that lies within an input spatial range [51]. Because of the lack of optimization of spatial predicates in existing graph query processors, Sun et al. proposed a query operator, GEOEXPAND, which adds spatial data awareness into a graphics DBMS to execute graph queries with spatial predicates efficiently [52]. Additionally, a carpool matching system was designed to recommend carpools based on vehicles' weekly frequent trajectories [53]. A time series of locations in a trajectory was connected while building a time tree through the use of the GraphAware Neo4j TimeTree library. For spatial data, the Neo4j Spatial library was used to model and query trajectory data. Using a carpool matching strategy, the efficiency and efficacy of the proposed system were evaluated [53]. Neo4j has also been applied in agriculture and animal husbandry applications, for example, using web technology and a Neo4j shell to evaluate the condition of the crops on the basis of geospatial data [54] and identifying relations between the members of a cattle herd based on spatial and graph databases [55].

Because Neo4j only supports the point geometry type and point-based spatial functions, it is generally used in location-related applications.

4.4. Apache Cassandra

Apache Cassandra is an open source, distributed, wide column storage database management system. Cassandra and the Cassandra Query Language (CQL) do not support spatial queries. Cassandra's main method for supporting geospatial data is Stratio's Lucene Index for Cassandra, a plugin for Apache Cassandra, which supports geospatial data indexing (point, lines, polygons, etc.), geospatial transformations (bounding box, centroid, convex hull, buffer, union, difference, intersection), and geospatial operations (intersects, contains, etc.). Lucene indexes, an extension of Cassandra secondary indexes, can be created through a CQL CREATE CUSTOM INDEX statement. The syntax is:

```
CREATE CUSTOM INDEX (IF NOT EXISTS)? <index_name>
 ON <table_name> ()
 USING 'com.stratio.cassandra.lucene.Index'
 WITH OPTIONS = <options>
```

In Cassandra, the main geometry objects include Point, LineString, and Polygon, while Cassandra itself does not provide a spatial index.

Cassandra is a highly scalable and high-performance data store, but it provides limited capabilities for data analyses and limited scalable functions, including a lack of adequate support for spatial data operations. To surmount these problems, research into an extension for the Cassandra Query Language was developed to implement spatial queries in the Cassandra database [56]. This extension research mainly converted the latitudinal/longitudinal values into a numeric geohash attribute and associated them with the data during data storage operations. Then, a spatial query parser and spatial syntax were designed and defined as a CQL spatial extension. After that, an aggregation algorithm was executed to reduce the search space and optimize the sub-queries sent to the cluster nodes. The stored data can be indexed through a geohashing technique [56]. Moreover, a novel approach used to couple Cassandra with Secondo DBMS was proposed by Nidzwetzki et al. to support all DBMS functions, including models of spatial and moving object data with high availability and scalability [57]. Using that work, Nidzwetzki et al. further expanded this approach [58] to build a DBMS that was distributed, general-purpose, fault-tolerant, and parallel. Similarly, to solve issues with spatial queries in the Cassandra database, a framework was developed by integrating of Hadoop and Cassandra for spatial query data stored in Cassandra [59]. The experimental results showed that a user-defined partitioning technique, called prefix-based partitioning, performed better in a geospatial search than did Cassandra's default partitioning algorithm [59].

For similar purposes, another framework combining Spark and Cassandra was proposed to provide data loading and data retrieval solutions for spatial data [60]. This framework includes a spatial data storage layer (based on Cassandra), a Spark core layer (using standard Spark core APIs), a spatial data processing layer (as an interface to query spatial and non-spatial data), and an application layer, in which a Spark-Cassandra connector [61] provides seamless integration between Spark and Cassandra [60].

Cassandra provides limited geometry types, geometry indexes, and functions, so extra design workings or tools/components are required in geospatial data processing, such as an index extension [56] and the combination with other tools [59–61].

4.5. Apache HBase

Apache HBase is an open source, distributed, versioned, and disk-based architecture database. HBase does not support a declarative query language, and queries in HBase are achieved through proprietary APIs. HBase does not have special geospatial functions to support geospatial data storage and querying [62]. However, researchers developed some methods and applications for processing geospatial data in HBase [63–70], such as a geographical database with geohash-based spatial indexes [63,71], big spatial data processing with Apache Spark [72], a geospatial data model [64], and a new spatial query method based on primary keys' indexing [73]. Additionally, an open source suite of tools, GeoMesa, was designed to implement large-scale geospatial analytics and querying in the cloud or in conjunction with the HBase and Cassandra databases [74].

Besides the above-mentioned research about geospatial processing in HBase, there has been other research related to geospatial processing. The first is based on the MapReduce mechanism [75,76]. Hadoop MapReduce is a software framework through which designers can easily write applications to process huge amounts of data in-parallel in large clusters of commodity hardware [77], while HBase uses base classes to support MapReduce jobs with HBase tables [76]. The central idea of MapReduce is that massive datasets are first divided into independent chunks stored in the clusters, and these chunks are processed by matching different tasks and methods in parallel [75,76].

Other researchers have designed a new index structure to manage data for HBase [69,70,78,79]. Du et al. proposed a novel hybrid index structure to organize data by developing a statistical grid-based R-Tree for indexing space and by using a Hilbert curve for neighbor finding [78]. In HBase, a novel spatial index structure with geohash encoding was designed [69,80]. Additionally, Jo et al. developed a hierarchical index structure for effective spatial query processing in HBase, called a Q-MBR (quadrant-based minimum bounding rectangle) tree [79]. Through Q-MBR, the space is split into

quadrants, and MBR is created in each quadrant. Then, the spatial objects are accessed through an index tree in a hierarchical manner. Based on the Q-MBR tree, different algorithms have been designed for different query operations [79].

The third research direction is to build storage models/schemas of spatial data in HBase [67,73,81–83]. Wang et al. proposed a Z storage schema with row keys based on a Z curve for massive spatial vector data in HBase [67]. After that, Zhang et al. improved this Z storage schema, and their experiments showed that the Z storage schema had a higher spatial query efficiency than did a tree-based storage schema (Quadtree storage and R-Tree storage schema) [83].

Furthermore, Zhai et al. combined the distributed HBase database and a global subdivision grid to manage data effectively: with their method, grid geocodes presented the spatial position of an object and were regarded as a key-value in HBase [81]. Meanwhile, Zheng et al. considered spatial adjacency and proposed a spatial data storage optimization strategy for the HBase database: their method stores adjacent spatial objects in the same data fragment [84].

Because HBase does no support geometry types, geometry indexes, or functions, extra design workings or tools/components are required to process geospatial data.

4.6. Redis

Redis is an open source (BSD licensed), in-memory key-value database that supports multiple data structures, including strings, hashes, lists, sets, bitmaps, geospatial indexes with radius queries, and streams. Redis implements queries through specific APIs and provides six geospatially related commands: geoadd, geodist, geohash, geopos, georadius, and georadiusbymember. These commands are easy to implement in geospatial data operations. For example, the geoadd command format is "GEOADD (set name) latitude longitude (object name)", and the user adds the specified geospatial object (latitude, longitude, name) to the specified key. Data are stored in the key as a sorted set, and in this way, the object can be retrieved using a query of the radius with the georadius or georadiusbymember commands.

An example is:

GEOADD building 15.45244 -76.78506 my-house

To delete a member from the Geo Set, Redis provides the ZREM command:

redis.zrem (building, my-house)

Redis is an in-memory, but persistent on disk database. When an important change in the data is generated, it is required to instruct Redis to save the change to disk. Additionally, Redis has a limited ability to create relationships between data objects.

Due to its characteristics, Redis is generally used for quick response tracking systems, such as ship tracking [16] and public transportation vehicle tracking [85], in which real-time data need to be displayed or processed in a timely manner, while the snapshot memory data are not required to be stored immediately.

4.7. Amazon DynamoDB

Amazon DynamoDB is a NoSQL database providing fast and predictable performance with seamless scalability, document storage, key-value storage, and low-level APIs (protocol-level interface) for managing database tables and indexes. In order to easily build location-based applications, the Geo Library for Amazon DynamoDB was designed, so that a GeoPoint (with a latitudinal value and a longitudinal value) is encoded in a GeoJSON string. Further, geohash indexes for fast location-based queries are used, including box queries and radius queries. Moreover, DynamoDB provides multiple geospatial operations functions, such as GeoPoint, putPoint, deletePoint, updatePoint, queryRectangle, and queryRadius. No related academic articles were found on geospatial processing based on research on DynamoDB.

4.8. Elasticsearch

Elasticsearch is a search engine and Document database that was developed in Java. Elasticsearch provides a query language, Domain Specific Language (DSL), based on JSON to define queries and supports. It also supports two Geo datatypes, including a Geo-point datatype (latitude and longitude pairs) and a Geo-shape datatype, which supports Points, Lines, Circles, Polygons, MultiPolygons, etc. In terms of queries, Elasticsearch has four functions: geo_shape, geo_bounding_box, geo_distance, and geo_polygon, and provides multiple PrefixTree, including GeohashPrefixTree and QuadPrefixTree. Additionally, geospatial data can be represented using either GeoJSON or Well-Known Text (WKT) format. An example of GeoJSON is shown here:

```
POST /example/_doc
{
"location" : {
 "type" : "point",
 "coordinates" : [-77.03653, 38.897676]
 }
}
```

We found some articles on Elasticsearch, including one on the usage of Elasticsearch for queries and the storage of local geographic information [86] and one on Elasticsearch as-a-service in Elastic's Elastic Cloud [87]. Elasticsearch is also used for indexing and querying big geospatial datasets in the GeoRocket system [36].

4.9. Splunk

Splunk Inc. has many software products with powerful search and analysis abilities for enterprise data management. One of them is Splunk Enterprise, which can fetch data from websites and the IoT (Internet of Things) and has excellent data management and mining performances. Splunk's query language is called the Splunk Process Language (SPL). In Splunk software, geospatial lookups should be used first to generate queries, and the query results are illustrated by a choropleth map visualization. A geospatial lookup maps event/object location coordinates in a geographic feature collection, called a Keyhole Markup Zipped (KMZ) file or a Keyhole Markup Language (KML) file. A format for creating a geospatial lookup is defined below:

```
[<lookup_name>]
external_type = geo
filename = <name_of_KMZ_file>
feature_id_element = <XPath_expression>
```

Splunk Enterprise provides two geospatial lookups for the United States and for other countries. No related academic article was found on geospatial processing based on Splunk applications and research.

4.10. Solr

Solr is an open-source search platform with high reliability, scalable indexing, search functions, fault tolerance, and load-balanced querying. Its default query parser is the "Lucene" query parser. To store spatial data, Solr supports the WKT and GeoJSON format, but the data format needs to be designated through a "format type name" before data can be stored. There are two inner parameters: f for the field name and w for the format name.

An example is:

```
geojson: [geo f=mySpatialField w=GeoJSON].
```

For geospatial data queries, Solr provides indexing points or other shapes, searching results from a bounding box or circle or other shapes, sorting research results in terms of distance, or even boosting results in terms of distance. Moreover, Solr supplies four main field types for spatial searches, including:

1. *LatLonPointSpatialField*: this is most commonly used for latitude–longitude point data;
2. *RptWithGeometrySpatialField*: for indexing and searching for non-point data (it can do points, as well, but it cannot do sorting/boosting);
3. *BBoxField*: for indexing bounding boxes, queries using a box, or specifying a search predicate (Intersects, Within, Contains, Disjoint, Equals);
4. *LatLonType* (now defunct): it still exists, but has been replaced by LatLonPointSpatialField.

Here, we give two examples for searching for a store within 5 km of a location (45.15, −93.85):

&q=:*&fq={!bbox sfield=store} & pt=45.15,-93.85 & d=5*
&q=:*&fq={!geofilt sfield=store} & pt=45.15,-93.85 & d=5*

The parameters mean the following:

d is the radial distance, usually in kilometers;
pt is the center point using the format "latitude, longitude";
sfield is a spatial index field;
the *geofilt* filter retrieves results based on the geospatial distance (circle distance) using a given point as the center of a circle and d as the radius;
the *bbox* (bounding box) filter uses the bounding box of the *geofilt* circle.

Because the bounding box is loose, some stores that are actually more than 5 km away may be found, but the *geofit* is accurate at 5 km.

As a back-end server, Solr has been used to index and search metadata services in the index node of the Earth System Grid Federation (ESGF), which can access distributed geospatial data [88]. In terms of geospatial data processing, Solr is widely used as a research engine, rather than for data storage [89,90], so combining it with other databases is required.

5. Comparisons of Geospatial Data Processing in NoSQL Databases

To compare 10 different NoSQL databases, their geospatial features are summarized in Table 4, which includes geometry primitives, the main geometry functions, spatial indexes, query language, and data format. In Table 4, it can be seen that nine out of the 10 databases support geospatial data and have special functions or procedures, except for HBase. However, perhaps due to the absence of geospatial features and geospatial functions in HBase, geospatial research is often done in the HBase database.

In terms of geometry objects, most NoSQL databases support multiple geometry objects, except for Amazon DynamoDB, HBase, Neo4j, and Redis. DynamoDB and Redis only support the point object, while the HBase database does not support any geometry objects. Neo4j also only supports the point geometry object, but an extended library, Neo4j Spatial, supports seven geometry objects. MongoDB, DynamoDB, and Elasticsearch support more comprehensive geometry functions than do other NoSQL databases and contain the operations of point, distance, and range. For spatial indexing, the geohash is the most common method and has been adopted in document databases, column-oriented database, and key-value databases. The tree structure is another common structure employed for indexing spatial data in NoSQL databases. In terms of query language, some NoSQL databases do not support a declarative query language, including MongoDB, HBase, Redis, and Amazon DynamoDB. They support REST queries and a proprietary API for building and issuing queries. In terms of supported geospatial data formats, eight out of 10 databases provide general GeoJSON or WKT data formats, except for HBase and Splunk.

In NoSQL databases, data models (including multi-models and search engines) can be classified into four major categories: key-value, graph model, wide column, and document stores. In fact, all of these data models can handle and manage geospatial data. However, different NoSQL databases can store and represent geospatial data in different ways according to the specific data model.

Graph databases are based on nodes (0D) and edges (1D), similar to the graph topology model relevant for spatial data. Two nodes and an edge (connection between the two nodes) can expediently represent two road crossings and a road between the two crossings in a topology relationship. However, graph databases do not support graph topological faces (2D), i.e., the space limited by an edged. Since Neo4j is a graph database, it can natively handle the 0D and 1D graph topological properties of geospatial data and can provide fast traversal operations [21].

In key-value mode (Redis), a historical building (building name) and this building's location (longitude and latitude), history, and construction information can also be easily stored in Redis (the building name is a key, and the other information is the stored values). The key-value database is an in-memory store where data loading and workload execution are incredibly fast [13]. Due to its in-memory store, the Redis database is generally applied in specific systems that require the illustration of data in real time and do not require the persistent storage of all data, such as in ship tracking [16].

However, because of the characteristics of these two data models, Neo4j and Redis mainly support the point geometry object. Therefore, key-value databases (Redis) [7,91] and graph databases (Neo4j) [53–55] only provide limited geospatial queries and functions, including distance calculations and location queries. Restricted geospatial data models hinder the applications for which key-value databases (Redis) and graph databases (Neo4j) are useful due to the complexity of geospatial data, especially that of polyline and polygon objects.

Document databases can differ in the details, but all document databases encode and encapsulate information into documents in a certain standard format. The common standard encoding formats include Extensible Markup Language (XML), JavaScript Object Notation (JSON), and Binary JSON (BSON). For geospatial data, the document databases use GeoJSON format, such as MongoDB, Couchbase, Amazon DynamoDB, and Elasticsearch.

Document databases have complex relationships with other NoSQL databases. For example, the search engine Elasticsearch provides ample operations for documents and is considered to be a document-oriented database. Additionally, a document database can sometimes be viewed as a key-value database, such as Redis, Couchbase, and Amazon DynamoDB. MongoDB is not a key-value database, but it uses the concept of key-value pairs, and documents are accessed using a key. Although document databases are intimately related to key-value databases, document-oriented databases, such as MongoDB, Couchbase, and Elasticsearch, process and manage geospatial data more effectively than do key-value databases [7]. This is mainly because document databases have more flexible queries for retrieving geospatial data than do key-value databases, including proximity queries and embedded topology analysis functions [17], and through the GeoJSON format, many document databases easily support or extend geospatial data management. Furthermore, document databases perform well in geospatial data queries [18,31], spatial data retrievals [33], and in terms of response times for loading big geospatial data [32]. MongoDB also has the best query time for node queries compared to Neo4j and PostgreSQL [92].

The wide column databases, Cassandra and HBase, store data tables in columns instead of in rows, and they are open-source, non-relational, distributed databases. HBase does not support geospatial processing; however, through MapReduce [75,76,93] and by designing new index structures [69,70,78,79] and storage models/schemas [67,73,81–83], the HBase database can now process geospatial data for different applications. As with Hbase, Cassandra can also use Hadoop MapReduce for geospatial data processing [59]. However, HBase is a column-oriented key-value data store, and Cassandra is essentially a hybrid between a key-value and a tabular database management system. Neither supplies a way to query by column and value, and query performance mainly depends on limited keys, so the column-family databases can be used efficiently for some special geospatial applications

that need simple geospatial queries, mass data insertion, and fast data retrieval [21]. Additionally, most research based on HBase has focused on vector spatial data [66,67,83,93,94], while document databases can handle raster data [7] and vector spatial data [36]. Because wide column databases do not have sufficient functions and queries to support geospatial data processing, another weakness of wide column databases (HBase) is that they require extra work for geospatial indexes and functions design [69,78,94]; this may cause limitations in the interoperability and sharing of designs compared to the indexes and functions of built-in databases. The spatial indexes and functions of the built-in database provide convenience and efficiency in design works, but fixed indexes and functions might limit their flexibility in some applications. Designers and developers must balance the convenience and flexibility of a design project, as well as considerations of the workload and complexity of the design.

Furthermore, geospatial indexing is vital for geospatial queries in NoSQL databases. For better query performance, some researchers have extended current indexing methods for different NoSQL databases, including R-Tree for MongoDB [44,46], geohash extensions for MongoDB [45,47], a graph-based expansion tree (GET) for Neo4j [95], and a new hybrid indexing scheme called HB+-trie for key-value storage [96].

Currently, document and wide column databases receive more academic attention than do graph and key-value databases in terms of geospatial data processing. The research on document databases has mainly concentrated on index improvement [44,45], performance analysis [18,23], and practical applications [38–40]. Additionally, due to the high performance of data insertion and retrieval in HBase, many researchers have designed systems and applications for geospatial data based on HBase [63,73,79,82,94].

A basic comparison of the geometry objects, main geometry functions, spatial indexes, and data formats supported by these NoSQL databases is shown in Table 4. A summary of geospatial data processing in different NoSQL databases (based on our literature review and analysis) is listed in Table 5. Of the NoSQL databases, document databases handle geospatial data processing the most effectively, considering the geometry objects they support, their data formats, their query performance, their geospatial functions, their index methods, and the amount of academic attention they receive. The other databases have their own advantages for specific scenarios.

Table 5. A summary of geospatial processing in NoSQL databases.

Data Models	Main Characteristics in Terms of Geospatial Processing	Main Applications	Academic Attention
Graph database	1. Fast graph traversal 2. Distance calculations and location query applications 3. Limited geospatial indexes, queries, and functions	Spatio-temporal data (moving object)	Low
Document databases	1. Good and stable performance in geospatial spatial data queries and retrievals 2. Wide application scenarios 3. Abundant and effective geospatial management, indexes, queries, and functions	Widely (distributed or web/cloud platform)	High
Wide column databases	1. Fast mass data insertion and data retrieval 2. Require users to design or construct indexes and query functions 3. Limited geospatial queries and functions, but provide basic classes' extension	Widely (distributed or web/cloud platform)	High

Table 5. *Cont.*

Data Models	Main Characteristics in Terms of Geospatial Processing	Main Applications	Academic Attention
Key-value database	1. Fast data loading and workloads execution 2. In-memory storage and specific application scenario 3. Limited geospatial queries and functions	Tracking applications	Low

6. Conclusions

In this paper, we summarized the state-of-the-art geospatial data processing used in the 10 most popular NoSQL databases and compared their performances based on geometry objects supported, geometry functions, spatial indexes, data formats, query languages, and use in academic research. Moreover, we analyzed the pros and cons of these NoSQL databases in geospatial data processing. Graph databases and key-value databases tend to express the geometry point object, without enough support for other geometric structures. This limits their geometric functions and applications. Moreover, these two types of databases have received little academic attention in terms of geospatial data processing. Document databases support a variety of geometric structures and provide a richer set of geospatial functions than do graph and key-value databases. Wide column databases only support a limited number of geospatial queries and functions; however, wide column databases have been adopted for many applications and have been studied extensively by academics, as have document databases. On the basis of our literature review, which included a systematic comparison of NoSQL database characteristics, we conclude that document databases are the best platform for geospatial data processing, as they load fast and have a good execution time, good query performance, and abundant geospatial functions and index methods. They have also received much academic attention.

Depending on the application scenario, graph databases, key-value databases, and wide column databases also have their own advantages. Additionally, geometry surface calculations and volume processing are not handled in the existing NoSQL databases. This could be a new direction for spatial processing research.

Author Contributions: Resources, methodology, and data collection, Dongming Guo; formal analysis and investigation, Dongming Guo and Erling Onstein; writing, original draft preparation, Dongming Guo; writing, review and editing, Dongming Guo and Erling Onstein. All authors have read and agreed to the published version of the manuscript.

Funding: NTNU Open Access publishing funds covered the article processing charges.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Vatsavai, R.R.; Ganguly, A.; Chandola, V.; Stefanidis, A.; Klasky, S.; Shekhar, S. Spatiotemporal data mining in the era of big spatial data: Algorithms and applications. In Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data, 6 November 2012; ACM: Redondo Beach, CA, USA, 2012; pp. 1–10.
2. Lee, J.-G.; Kang, M. Geospatial Big Data: Challenges and Opportunities. *Big Data Res.* **2015**, *2*, 74–81. [CrossRef]
3. Liu, Z.; Guo, H.; Wang, C. Considerations on Geospatial Big Data. *IOP Conf. Ser.* **2016**, *46*, 012058. [CrossRef]
4. Yue, P.; Jiang, L.C. BigGIS: How Big Data Can Shape Next-Generation GIS. In Proceedings of the Third International Conference on Agro-Geoinformatics, Beijing, China, 11–14 August 2014; pp. 413–418.
5. Baralis, E.; Valle, A.D.; Garza, P.; Rossi, C.; Scullino, F. SQL versus NoSQL Databases for Geospatial Applications. In Proceedings of the 2017 IEEE International Conference on Big Data, IEEE, Boston, MA, USA, 11–14 December 2017; pp. 3388–3397.

6. Schmid, S.; Galicz, E.; Reinhardt, W. Performance investigation of selected SQL and NoSQL databases. In Proceedings of the AGILE 2015, Lisbon, Portugal, 9–12 June 2015.
7. Hu, F.; Xu, M.C.; Yang, J.C.; Liang, Y.S.; Cui, K.J.; Little, M.M.; Lynnes, C.S.; Duffy, D.Q.; Yang, C.W. Evaluating the Open Source Data Containers for Handling Big Geospatial Raster Data. *ISPRS Int. J. Geo Inf.* **2018**, *7*, 144.
8. Obe, R.O.; Hsu, L.S. *PostGIS in Action*, 2nd ed.; Manning Publications Co.: Shelter Island, NY, USA, 2015.
9. Zhong, Y.; Han, J.; Zhang, T.; Fang, J. A distributed geospatial data storage and processing framework for large-scale WebGIS. In Proceedings of the 2012 20th International Conference on Geoinformatics, Hongkong, China, 15–17 June 2012; pp. 1–7.
10. Oracle Database Documentation. Available online: https://docs.oracle.com/en/database/oracle/oracle-database/index.html (accessed on 4 October 2019).
11. Au, C.; Rischpater, R. Geospatial Data with Azure SQL Database. In *Microsoft Mapping: Geospatial Development in Windows 10 with Bing Maps and C#*; Apress: Berkeley, CA, USA, 2015; pp. 33–53.
12. Microsoft SQL Server/Geospatial Data. Available online: https://en.wikibooks.org/wiki/Microsoft_SQL_Server/Geospatial_Data (accessed on 8 May 2018).
13. Tang, E.Q.; Fan, Y.S. Performance Comparison between Five NoSQL Databases. In Proceedings of the 2016 7th International Conference on Cloud Computing and Big Data, Macau, China, 16–18 November 2016; pp. 105–109.
14. Cheng, B.; Guan, X. Design and evaluation of a high-concurrency web map tile service framework on a high performance cluster. *Int. J. Grid Distrib. Comput.* **2016**, *9*, 127–142. [CrossRef]
15. Ramzan, S.; Bajwa, I.S.; Kazmi, R. Amna. Challenges in NoSQL-Based Distributed Data Storage: A Systematic Literature Review. *Electronics* **2019**, *8*, 29. [CrossRef]
16. Akbar, A.M.A.M.; Purnama, I.K.E.; Nugroho, S.M.S.; Hariadi, M. Fast and Efficient Cluster Based Map for Ship Tracking. In Proceedings of the 2018 International Conference on Computer Engineering, Network and Intelligent Multimedia, Surabaya, Indonesia, 26–27 November 2018; pp. 265–269.
17. Polakova, M.; Vitols, G. Use of NoSQL Technology for Analysis of Unstructured Spatial Data. In *Research for Rural Development 2018*; Treija, S., Skujeniece, S., Eds.; Latvia University of Life Sciences and Technologies: Yelgava, Latvia, 2018; Volume 2, pp. 267–270.
18. Agarwal, S.; Rajan, K.S. Performance analysis of MongoDB versus PostGIS/PostGreSQL databases for line intersection and point containment spatial queries. *Spat. Inf. Res.* **2016**, *24*, 671–677. [CrossRef]
19. Agarwal, S.; Rajan, K. Analyzing the performance of NoSQL vs. SQL databases for Spatial and Aggregate queries. In Proceedings of the Free and Open Source Software for Geospatial (FOSS4G) Conference Proceedings, Boston, MA, USA, 14–19 August 2017.
20. Simmonds, R.; Watson, P.; Halliday, J. Antares: A Scalable, Real-time, Fault tolerant Data Store for Spatial Analysis. In *2015 IEEE World Congress on Services (SERVICES)*; Zhang, L.J., Bahsoon, R., Eds.; IEEE: New York City, NY, USA, 2015; pp. 105–112.
21. Amirian, P.; Basiri, A.; Winstanley, A. Evaluation of Data Management Systems for Geospatial Big Data. In *Computational Science and Its Applications—ICCSA 2014*; Springer International Publishing: Guimarães, Portugal, 2014; pp. 678–690.
22. Brink, L.V.D.; Barnaghi, P.; Tandy, J.; Atemezing, G.; Atkinson, R.; Cochrane, B.; Fathy, Y.; Castro, R.G.; Haller, A.; Harth, A.; et al. Best practices for publishing, retrieving, and using spatial data on the web. *Semant. Web* **2019**, *10*, 95–114. [CrossRef]
23. More, N.P.; Nikam, V.B.; Sen, S.S. Experimental Survey of Geospatial Big Data Platforms. In Proceedings of the 2018 Ieee 25th International Conference on High Performance Computing Workshop, Bengaluru, India, 17–20 December 2018; pp. 137–143.
24. Patroumpas, K.; Giannopoulos, G.; Athanasiou, S. Towards GeoSpatial semantic data management: Strengths, weaknesses, and challenges ahead. In Proceedings of the ACM International Symposium on Advances in Geographic Information Systems, Association for Computing Machinery, New York, NY, USA, 7–10 August 2017; pp. 301–310.
25. Consortium, O.G. OpenGIS®Implementation Standard for Geographic Information—Simple Feature Access—Part 1: Common Architecture. Available online: https://www.ogc.org/standards/sfa (accessed on 4 December 2011).
26. Egenhofer, M.J.; Franzosa, R.D. Point-set topological spatial relations. *Int. J. Geogr. Inf. Syst.* **1991**, *5*, 161–174. [CrossRef]

27. Randell, D.A.; Cui, Z.; Cohn, A.G. A spatial logic based on regions and connection. In Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning, Cambridge, MA, USA, 23–25 October 1992; pp. 165–176.
28. Battle, R.; Kolas, D. Enabling the geospatial Semantic Web with Parliament and GeoSPARQL. *Semant. Web* **2012**, *3*, 355–370. [CrossRef]
29. Qian, C.Y.; Yi, C.; Cheng, C.Q.; Pu, G.L.; Wei, X.F.; Zhang, H.C. GeoSOT-Based Spatiotemporal Index of Massive Trajectory Data. *ISPRS Int. J. Geo Inf.* **2019**, *8*, 284. [CrossRef]
30. DB-Engines Ranking. Available online: https://db-engines.com/en/ranking (accessed on 8 October 2019).
31. Bartoszewski, D.; Piorkowski, A.; Lupa, M. The Comparison of Processing Efficiency of Spatial Data for PostGIS and MongoDB Databases, in Beyond Databases. In *Architectures and Structures. Paving the Road to Smart Data Processing and Analysis*; Kozielski, S., Mrozek, D., Kasprowski, P., Małysiak-Mrozek, B., Kostrzewa, D., Eds.; Springer: Cham, Switzerland, 2019; pp. 291–302.
32. Laksono, D. Testing Spatial Data Deliverance in SQL and NoSQL Database Using NodeJS Fullstack Web App. In Proceedings of the 2018 4th International Conference on Science and Technology, ICST 2018, Yogyakarta, Indonesia, 7–8 August 2018.
33. Duan, M.R.; Chen, G. Assessment of MongoDB's Spatial Retrieval Performance. In Proceedings of the 2015 23rd International Conference on Geoinformatics, Wuhan, China, 19–21 June 2015.
34. Bordogna, G.; Ciriello, D.E.; Psaila, G. Flexible Framework to Cross-Analyze Heterogeneous Multi-Source Geo-referenced Information: The J-CO-QL Proposal and its Implementation. In Proceedings of the 2017 IEEE/Wic/Acm International Conference on Web Intelligence, Association for Computing Machinery, Leipzig, Germany, 23–26 August 2017; pp. 499–508.
35. Detti, A.; Rossi, G.; Melazzi, N.B. Exploiting Information-Centric Networking to Federate Spatial Databases. *IEEE Access* **2019**, *7*, 165248–165261. [CrossRef]
36. Krämer, M. GeoRocket: A scalable and cloud-based data store for big geospatial files. *SoftwareX* **2020**, *11*, 100409. [CrossRef]
37. Rainho, F.D.; Bernardino, J. Web GIS: A new system to store spatial data using GeoJSON in MongoDB. In Proceedings of the 2018 13th Iberian Conference on Information Systems and Technologies, Caceres, Spain, 13–16 June 2018.
38. Yaqot, M.; Trigunarsyah, B. Web-GIS to support maintenance reporting system: Application in Saudi Arabia. *Infrastruct. Asset Manag.* **2018**, *5*, 14–21. [CrossRef]
39. Zhang, X.M.; Song, W.; Liu, L.M. An Implementation Approach to Store GIS Spatial Data on NoSQL Database. In Proceedings of the 2014 22nd International Conference on Geoinformatics, Kaohsiung, Taiwan, 25–27 July 2014.
40. Zhao, C.X.; Wu, A.H.; Tao, Y.C.; Gao, W.C.; Liu, J. Design and implementation of Beijing fundamental geospatial framework platform. In Proceedings of the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Suzhou, China, 14–16 May 2014.
41. Ferro, M.; Fragoso, R.; Fidalgo, R. Document-oriented geospatial data warehouse: An experimental evaluation of SOLAP queries. In Proceedings of the 21st IEEE Conference on Business Informatics, CBI 2019, Moscow, Russia, 15–17 July 2019.
42. Ferro, M.; Lima, R.; Fidalgo, R. Evaluating redundancy and partitioning of geospatial data in document-toriented data warehouses. In *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer: Genève, Switzerland, 2019; pp. 221–235.
43. Jun, S.; Lee, S. Prototype system for geospatial data building-sharing developed by utilizing open source web technology. *Spat. Inf. Res.* **2017**, *25*, 725–733. [CrossRef]
44. Xiang, L.G.; Shao, X.T.; Wang, D.H. Providing R-Tree Support for MongoDB. *ISPRS Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2016**, *XLI-B4*, 545–549. [CrossRef]
45. Li, Y.; Kim, D.; Shin, B.S. Geohashed Spatial Index Method for a Location-Aware WBAN Data Monitoring System Based on NoSQL. *J. Inf. Process. Syst.* **2016**, *12*, 263–274.
46. Xiang, L.G.; Huang, J.T.; Shao, X.T.; Wang, D.H. A MongoDB-Based Management of Planar Spatial Data with a Flattened R-Tree. *ISPRS Int. J. Geo Inf.* **2016**, *5*, 119. [CrossRef]
47. Guan, X.F.; Bo, C.; Li, Z.Q.; Yu, Y.J. ST-Hash: An Efficient Spatiotemporal Index for Massive Trajectory Data in a NoSQL Database. In Proceedings of the 2017 25th International Conference on Geoinformatics, Buffalo, NY, USA, 2–4 August 2017.

48. Why Couchbase? Available online: https://docs.couchbase.com/server/6.0/introduction/intro.html (accessed on 10 June 2019).
49. Spatial Values. Available online: https://neo4j.com/docs/cypher-manual/3.5/syntax/spatial/ (accessed on 2 September 2019).
50. Neo4j Spatial v0.24-neo4j-3.1.4. Available online: https://neo4j-contrib.github.io/spatial/0.24-neo4j-3.1/index.html (accessed on 1 June 2017).
51. Sarwat, M.; Sun, Y.H.; IEEE. Answering Location-Aware Graph Reachability Queries on GeoSocial Data. In Proceedings of the 2017 IEEE 33rd International Conference on Data Engineering, San Diego, CA, USA, 19–22 April 2017; pp. 207–210.
52. Sun, Y.H.; Sarwat, M. A spatially-pruned vertex expansion operator in the Neo4j graph database system. *Geoinformatica* **2019**, *23*, 397–423. [CrossRef]
53. Bakkal, F.; Savas, N.S.; Eken, S.; Sayar, A. Modeling and Querying Trajectories using Neo4j Spatial and TimeTree for Carpool Matching. In Proceedings of the 2017 Ieee International Conference on Innovations in Intelligent Systems and Applications, Gdynia, Poland, 3–5 July 2017; pp. 219–222.
54. Idziaszek, P.; Mueller, W.; Gorna, K.; Okon, P.; Boniecki, P.; Koszela, K.; Fojud, A. Identification of the condition of crops based on geospatial data embedded in graph databases. In Proceedings of the Ninth International Conference on Digital Image Processing, Hong Kong, China, 19–22 May 2017.
55. Mueller, W.; Rudowicz-Nawrocka, J.; Otrzasek, J.; Idziaszek, P.; Weres, J. Spatial data and graph databases for identifying relations among members of cattle herd. *Inform. Geoinf. Remote Sens. Conf. Proc. Sgem* **2016**, *1*, 835–842.
56. Ben Brahim, M.; Drira, W.; Filali, F.; Hamdi, N. Spatial data extension for Cassandra NoSQL database. *J. Big Data* **2016**, *3*, 11. [CrossRef]
57. Nidzwetzki, J.K.; Guting, R.H. Distributed SECONDO: A Highly Available and Scalable System for Spatial Data Processing. In *Advances in Spatial and Temporal Databases*; Claramunt, C., Schneider, M., Wong, R.C.-W., Xiong, L., Loh, W.-K., Shahabi, C., Li, K.J., Eds.; Springer: Genève, Switzerland, 2015; pp. 491–496.
58. Nidzwetzki, J.K.; Guting, R.H. DISTRIBUTED SECONDO: An extensible and scalable database management system. *Distrib. Parallel Databases* **2017**, *35*, 197–248. [CrossRef]
59. Vasavi, S.; Priya, M.P.; Gokhale, A.A. Framework for geospatial query processing by integrating cassandra with hadoop. In *Knowledge Computing and Its Applications: Knowledge Manipulation and Processing Techniques: Volume 1*; Springer: Singapore, 2018; pp. 131–160.
60. Shah, P.; Chaudhary, S. Big data analytics framework for spatial data. In *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer: Genève, Switzerland, 2018; pp. 250–265.
61. Website of Spark-Cassandra-Connector. Available online: https://github.com/datastax/spark-cassandra-connector (accessed on 4 January 2020).
62. Apache HBase ™ Reference Guide. Available online: https://hbase.apache.org/book.html (accessed on 16 April 2020).
63. Le, H.V.; Takasu, A. G-HBase: A High Performance Geographical Database Based on HBase. *Ieice Trans. Inf. Syst.* **2018**, *101*, 1053–1065. [CrossRef]
64. Han, D.; Stroulia, E. HGrid: A Data Model for Large Geospatial Data Sets in HBase. In Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing, Santa Clara, CA, USA, 28 June–3 July 2013; pp. 910–917.
65. Han, D.; Stroulia, E. Federating Web-Based Applications on a Hierarchical Cloud. In Proceedings of the 2014 IEEE 7th International Conference on Cloud Computing, Anchorage, AK, USA, 27 June–2 July 2014; IEEE: Anchorage, AK, USA, 2014; pp. 946–947.
66. Zhang, N.Y.; Zheng, G.Z.; Chen, H.J.; Chen, J.Y.; Chen, X. HBaseSpatial: A Scalable Spatial Data Storage Based on HBase. In Proceedings of the 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications, Beijing, China, 24–26 September 2014; pp. 644–651.
67. Wang, Y.; Li, C.J.; Li, M.; Liu, Z.L. HBase storage schemas for massive spatial vector data. *Clust. Comput. J. Netw. Softw. Tools Appl.* **2017**, *20*, 3657–3666. [CrossRef]
68. Jo, B.; Jung, S. Quadrant-Based Minimum Bounding Rectangle-Tree Indexing Method for Similarity Queries over Big Spatial Data in HBase. *Sensors* **2018**, *18*, 3032. [CrossRef]

69. He, S.W.; Chu, L.X.; Li, X.Y. Spatial Query Processing for Location Based Application on Hbase. In Proceedings of the 2017 IEEE 2nd International Conference on Big Data Analysis, Beijing, China, 10–12 March 2017; pp. 115–119.
70. Chen, X.Y.; Zhang, C.; Ge, B.; Xiao, W.D. Spatio-temporal Queries in HBase. In Proceedings of the 2015 IEEE International Conference on Big Data, Santa Clara, CA, USA, 29 October–1 November 2015; pp. 1929–1937.
71. Li, L.H.; Liu, W.D.; Zhong, Z.Y.; Huang, C.Q. SP-Phoenix: A Massive Spatial Point Data Management System based on Phoenix. In Proceedings of the IEEE 20th International Conference on High Performance Computing and Communications/IEEE 16th International Conference on Smart City/IEEE 4th International Conference on Data Science and Systems, Exeter, UK, 28–30 June 2018; pp. 1634–1641.
72. Shangguan, B.; Yue, P.; Wu, Z.; Jiang, L. Big spatial data processing with Apache Spark. In Proceedings of the 2017 6th International Conference on Agro-Geoinformatics, Agro-Geoinformatics, Fairfax, VA, USA, 7–10 August 2017.
73. Zheng, K.; Zheng, K.; Fang, F.L.; Zhang, M.; Li, Q.; Wang, Y.H.; Zhao, W.Y. An extra spatial hierarchical schema in key-value store. *Clust. Comput. J. Netw. Softw. Tools Appl.* **2019**, *22*, S6483–S6497. [CrossRef]
74. GeoMesa: Store, Index, Query, and Transform Spatio-Temporal Data at Scale in HBase, Accumulo, Cassandra, Kafka and Spark. Available online: https://www.geomesa.org/index.html (accessed on 16 August 2018).
75. Cho, W.; Choi, E. A basis of spatial big data analysis with map-matching system. *Clust. Comput. J. Netw. Softw. Tools Appl.* **2017**, *20*, 2177–2192. [CrossRef]
76. Gao, F.; Yue, P.; Wu, Z.; Zhang, M. Geospatial data storage based on HBase and MapReduce. In Proceedings of the 2017 6th International Conference on Agro-Geoinformatics, Fairfax, VA, USA, 7–10 August 2017; pp. 55–58.
77. MapReduce Tutorial. Available online: http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html (accessed on 22 August 2019).
78. Du, N.B.; Zhan, J.F.; Zhao, M.; Xiao, D.R.; Xie, Y.C. Spatio-temporal Data Index model of moving objects on fixed networks using HBase. In Proceedings of the 2015 Ieee International Conference on Computational Intelligence and Communication Technology Cict 2015, Ghaziabad, India, 13–14 February 2015; pp. 247–251.
79. Jo, B.; Jung, S. Quadrant-Based MBR-Tree Indexing Technique for Range Query Over HBase. In Proceedings of the 7th International Conference on Emerging Databases: Technologies, Applications, and Theory, Busan, Korea, 7–9 August 2017; Lee, W., Choi, W., Jung, S., Song, M., Eds.; Springer: Singapore, 2018; pp. 14–24.
80. Van, L.H.; Takasu, A. An Efficient Distributed Index for Geospatial Databases. In Proceedings of the International Conference on Database and Expert Systems Applications, Valencia, Spain, 1–4 September 2015; pp. 28–42.
81. Zhai, W.X.; Yang, Z.; Wang, L.; Wu, F.L.; Cheng, C.Q. The Non-sql Spatial Data Management Model in Big Data Time. In Proceedings of the 2015 IEEE International Geoscience and Remote Sensing Symposium, Milan, Italy, 26–31 July 2015; pp. 4506–4509.
82. Wang, K.; Liu, G.L.; Zhai, M.; Wang, Z.W.; Zhou, C.Y. Building an efficient storage model of spatial-temporal information based on HBase. *J. Spat. Sci.* **2019**, *64*, 301–317. [CrossRef]
83. Zhang, D.F.; Wang, Y.; Liu, Z.L.; Dai, S.J. Improving NoSQL Storage Schema Based on Z-Curve for Spatial Vector Data. *IEEE Access* **2019**, *7*, 78817–78829. [CrossRef]
84. Zheng, K.; Gu, D.; Fang, F.; Zhang, M.; Zheng, K.; Li, Q. Data storage optimization strategy in distributed column-oriented database by considering spatial adjacency. *Clust. Comput. J. Netw. Softw. Tools Appl.* **2017**, *20*, 2833–2844. [CrossRef]
85. Thomas, G.; Alexander, G.; Sasi, P.M. Design of High Performance Cluster based Map for Vehicle Tracking of public transport vehicles in Smart City. In Proceedings of the 2017 IEEE Region 10 International Symposium on Technologies for Smart Cities, Cochin, India, 14–16 July 2017.
86. Bartlett, R. Local geographic information storing and querying using elasticsearch. In Proceedings of the 13th Workshop on Geographic Information Retrieval, Lyon, France, 28–29 November 2019.
87. Toepke, S.L. Leveraging elasticsearch and botometer to explore volunteered geographic information. In Proceedings of the International ISCRAM Conference, Rochester, NY, USA, 20–23 May 2018.
88. Cinquini, L.; Crichton, D.; Mattmann, C.; Harney, J.; Shipman, G.; Wang, F.Y.; Ananthakrishnan, R.; Miller, N.; Denvil, S.; Morgan, M.; et al. The Earth System Grid Federation: An open infrastructure for access to distributed geospatial data. *Future Gener. Comput. Syst. Int. J. Grid Comput. Escience* **2014**, *36*, 400–417. [CrossRef]

89. Florance, P.; McGee, M.; Barnett, C.; McDonald, S. The Open Geoportal Federation. *J. Map Geogr. Libr.* **2015**, *11*, 376–394. [CrossRef]
90. Corti, P.; Kralidis, A.T.; Lewis, B. Enhancing discovery in spatial data infrastructures using a search engine. *Peerj Comput. Sci.* **2018**, *4*, e152. [CrossRef]
91. Celko, J. *Complete Guide to NoSQL, What Every SQL Professional Needs to Know about Non-Relational Databases*; Morgan Kaufmann: Boston, MA, USA, 2014; pp. 103–117.
92. Sharma, M.; Sharma, V.D.; Bundele, M.M. Performance Analysis of RDBMS and No SQL Databases: PostgreSQL, MongoDB and Neo4j. In Proceedings of the 3rd International Conference and Workshops on Recent Advances and Innovations in Engineering, ICRAIE 2018, Jaipur, India, 22–25 November 2018.
93. Wang, L.; Chen, B.; Liu, Y.H. Distributed Storage and Index of Vector Spatial Data Based on HBase. In Proceedings of the 2013 21st International Conference on Geoinformatics, Kaifeng, China, 20–22 June 2013.
94. Jiang, H.; Kang, J.F.; Du, Z.H.; Zhang, F.; Huang, X.Z.; Liu, R.Y.; Zhang, X.T. Vector Spatial Big Data Storage and Optimized Query Based on the Multi-Level Hilbert Grid Index in HBase. *Information* **2018**, *9*, 116. [CrossRef]
95. Zhang, H.C.; Lu, F.; Chen, J. A Line Graph-Based Continuous Range Query Method for Moving Objects in Networks. *ISPRS Int. J. Geo Inf.* **2016**, *5*, 246. [CrossRef]
96. Ahn, J.S.; Seo, C.; Mayuram, R.; Yaseen, R.; Kim, J.S.; Maeng, S. ForestDB: A Fast Key-Value Storage System for Variable-Length String Keys. *IEEE Trans. Comput.* **2016**, *65*, 902–915. [CrossRef]