ON THE EFFICIENCY OF RELATIONAL, DOCUMENT AND GRAPH DATA MODELS FOR MANAGING MOBILE SPATIAL DATA

PEDRO ONOFRE SANTOS

EFICIÊNCIA DOS MODELOS DE DADOS RELACIONAL, DOCUMENTO E GRAFO PARA O GERENCIAMENTO DE DADOS GEOGRÁFICOS MÓVEIS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Mirella Moura Moro
Coorientador: Clodoveu Augusto Davis Jr.

Belo Horizonte

Março de 2015

PEDRO ONOFRE SANTOS

ON THE EFFICIENCY OF RELATIONAL, DOCUMENT AND GRAPH DATA MODELS FOR MANAGING MOBILE SPATIAL DATA

Dissertation presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Mirella Moura Moro
Co-Advisor: Clodoveu Augusto Davis Jr.

Belo Horizonte
March 2015

Santos, Pedro Onofre

S237e On the efficiency of relational, document and graph data models for managing mobile spatial data / Pedro Onofre Santos. — Belo Horizonte, 2015
xxii, 72 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientadora: Mirella Moura Moro
Coorientador: Clodoveu Augusto Davis Jr.

1. Computação – Teses 2. Benchmarking (Administração) - Teses. 3. Avaliação de desempenho – Teses. 4. Sistemas de informação geográfica – Teses. I. Orientadora. II. Coorientador. III. Título.

CDU 519.6*75(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

On the efficiency of relational, document and graph data models for managing mobile spatial data

PEDRO ONOFRE SANTOS

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROFA. MIRELLA MOURA MORO - Orientadora
Departamento de Ciência da Computação - UFMG

PROF. CLODOVEU AUGUSTO DAVIS JÚNIOR
Departamento de Ciência da Computação - UFMG

PROF. RODRYGO LUIS TEODORO SANTOS
Departamento de Ciência da Computação - UFMG

PROF. JUGURTA LISBOA FILHO
Departamento de Informática - UFV

Belo Horizonte, 26 de março de 2015.

Acknowledgments

My most sincere thanks to Mirella and Clodoveu for their knowledge, guidance, patience and willingness to help me in any way. This work wouldn't be nearly as possible without them.

I also want to thank Ruth for her kindness, understanding, patience and support, she is the best person in the world.

My thanks to the peers and colleagues which helped so much during my studies: Bruno dos Santos Azevedo Cardoso, Cássia do Carmo Vieira, Tarcísio Guerra Savino Filó, Harlley Augusto de Lima, Daniel Hasan Dalip, Keiller Nogueira, Rodrigo Augusto da Silva Alves and Wladston Viana Ferreira Filho.

Also, I want to thank all my old Eletra's friends, whose friendship endured for the last ten years: Amanda, Burns, Cabeça, Cuia, Denis, Denise, Emo, Hudson, Jana, Junca, Luiz, Lula, Madruga, Pardal, Peão, Rafa, Sabará, Samsam, Sidoca, Sir and Thi. In that list, not mutually exclusive, but I want to thank our Dota 2 buddies which helped our stress levels rise and our win rate fall: Bill, Brunasso, Cuia, Daleste, Giuli, Jovem, Lula, Palhaço, Pardal, Sabará, Sir, Tutor and Vjunca.

I also want to thank my parents for letting me live rent free for 25 years and my grandmas for the much needed meals and desserts. If I have forgotten your name, know that this work would not be possible without all the people in my life, as they are too many to remember, I'm sorry.

Lastly, I want to thank God who has always been there for me.

This work is supported by a FAPEMIG scholarship. This financial support is gratefully acknowledged.

"What is better - to be born good, or to overcome your evil nature through great effort?"

(Paarthurnax)

Resumo

Vários dos sistemas de informação atuais apresentam volume de dados crescente combinado com diversidade e mobilidade de suas aplicações. Entretanto, selecionar uma infraestrutura computacional apropriada é ainda um grande desafio para projetistas de tais sistemas. Esta dissertação demonstra como técnicas de avaliação de desempenho atuais podem não ser ideais para comparar o desempenho de sistemas de bancos de dados espaciais. Particularmente, considera-se as necessidades de usuários móveis, as quais incluem tráfego constante de dados espaciais, tais como consulta por pontos de interesse, visualização de mapas, zoom e caminhamento, roteamento e rastreamento de localização. A avaliação realizada mostra que para obter uma comparação justa é necessário utilizar cargas específicas (de dados e consultas) para cada característica móvel - o que não é atualmente obtido através das ferramentas de benchmark presentes no mercado. São também comparadas tecnologias de sistemas de dados relacionais, orientados a documentos e baseados em grafos (NoSQL). De modo geral, este estudo demonstra que as metodologias genéricas de benchmark não são ótimas e podem levar a um projeto físico longe do ideal.

Palavras-chave: Análise de Desempenho, Bancos de Dados Geográficos, NoSQL, Big Data.

Abstract

Increasing data volume, application diversity and mobility are the foremost characteristics of many current information systems. However, selecting the appropriate computational infrastructure is still a hard task for the designers of such systems. This work demonstrates how current performance evaluation techniques may not be ideal when comparing the performance of spatial database management systems. Specifically, we consider the needs of mobile users that involve constant spatial data traffic, such as querying for points of interest, map visualization, zooming and panning, routing and location tracking. Our evaluation shows that a fair comparison requires specific workloads for each mobile feature – which is not currently achievable by the industry's standard benchmark tools. We then compare technologies in relational (SQL), document-oriented and graph-based DBMSs (NoSQL). Overall, this study demonstrates that the one-size-fits-all benchmark methodologies are not optimal and may lead to a far from an ideal system design.

Keywords: Benchmark, Spatial Databases, NoSQL, Big Data.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

The gathering and storage of spatial data regarding mineral resources, properties, water sources and other landscape attributes were always important tasks of organized human societies [14]. This however has changed greatly with the advance of computer systems capable of handling such data, which was previously limited to paper-written documents and maps. Such computer systems capable of processing and storing spatial attributes are called *Geographic Information Systems* (GIS). Traditional examples of GIS include: GRASS GIS[1], QGIS[2] and ArcGIS[3].

Geographic information systems that provide public services often manage very large databases with lots of user requests. Such GIS applications require continuous availability, low response time and high throughput. Also, the underlying geographic databases deal with spatial objects with varying complexity, for which operations are much more expensive than in relational database management systems (RDBMS) that manage conventional data [15, 22, 32, 49, 51].

Specifically, the demand for GIS and spatially-enhanced applications, mostly for mobile devices, has increased alongside the user base. For example, in Brazil, the number of mobile devices grows rapidly and, according to the Brazilian Institute of Geography and Statistics (IBGE)[4], it now reaches over 75% of the Brazilian population. Such behavior reflects a pattern already present in the United States and Europe, making DBMS development a continuous task in order to excel upcoming challenges[5].

[1] GRASS GIS: `http://grass.osgeo.org/`

[2] QGIS project: `http://www.qgis.org`

[3] ArcGIS: `http://www.arcgis.com/`

[4] IBGE: `http://www.ibge.gov.br/home/estatistica/pesquisas/pesquisa_resultados.php?id_pesquisa=40`

[5] MobiThinking Compendium of Mobile Statistics: `http://mobiforge.com/research-analysis/global-mobile-statistics-2014-home-all-latest-stats-mobile-web-apps-marketing-advertising-subscriber`

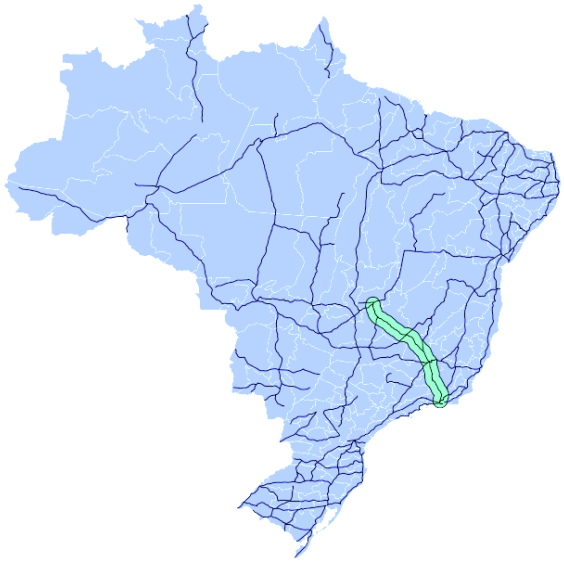


Figure 1.1: Brazilian main highways in blue with a spatial buffer for BR-040 highlighted.

In this context, geographic location resources such as integrated Global Positioning System (GPS) modules made spatially-aware applications more attractive and useful. At the same time, such applications are more frequently used, reaching a point where most features accessed by the user contain geographic data and metadata.

Examples of spatially enhanced applications and geographic information systems in which geographic data is uploaded along with conventional data and metadata include Google Maps[6], Foursquare[7] and Waze[8]. The spatial abilities of those applications make them so useful for so many people that a multi billion-dollar industry has been formed around them. Hence, the application's performance, reliability and scalability play very important roles in the user's decision to acquire and use such tools in a daily routine.

From the perspective of design and development of the databases underlying such applications, new requirements are becoming important. The trend towards novel management tools and techniques that replace Relational DBMS reflects such requirements, particularly for big data [48]. However, it is still not clear how to select an adequate spatial data management tool, considering the requirements of current applications, specially mobile ones. Such decision is far from simple, as current work shows incredible differences on evaluating queries over general and spatial big data (e.g., [21, 36, 49, 51] among many others).

[6]Google Maps: `http://maps.google.com`
[7]Foursquare: `http://www.foursquare.com`
[8]Waze: `http://www.waze.com`

Figure 1.2: Nearby points of interest searched by the user.

Spatial data is a multidimensional, more complex structure with slower processing time than primitive data types, such as bit, byte, integer, float as well as string, date and many others that are implemented in most DBMS. In a relational database, querying for data in a single table is straightforward in terms of query planning and hardware resource allocation. On the other hand, when performing spatial functions (mainly spatial joins), large amounts of memory are requested by the query, much more than the usual number/string processing, which leads to a large volume of allocated memory. Furthermore, the density of spatial objects on the dataset, the size of each object and even their shape can influence in the query performance.

For example, Figure 1.1 presents a map with Brazilian highways and highlights a spatial buffer created around highways Presidente Juscelino Kubitschek and Washington Luís. These two highways form BR-040, which connects Brazil's capital Brasília (DF) to Rio de Janeiro (RJ), one of the heaviest traffic highways in Brazil.

The density of spatial objects also plays an important role, as the queries for the nearby points of interest rely heavily on CPU processing power, alongside with the efficiency of spatial indexes. Figure 1.2 illustrates the result of a query for points of interest located near the user's position in the center of the map.

In this dissertation, we introduce a new methodology for comparing spatial DBMSs. Specifically, we focus on three different database models: relational, document-oriented and graph based. We then evaluate their performance by using one representative implementation of each model. Then, we experimentally evaluate the performance of various DBMSs with spatial features, in order to demonstrate the wide range of behavior of such tools under load.

Table 1.1: Comparison between this study and previous related work.

Study	SQL	NoSQL	Real Workload	Spatial Data	Big Data
Spatial Star Schema Benchmark [40]	✓	?	×	✓	×
TPC-C [10]	✓	×	×	×	×
BigBench [19]	✓	×	×	×	✓
Jackpine [43]	✓	×	✓	✓	×
Our study	✓	✓	✓	✓	✓

More over, our evaluation differs from other studies in three crucial ways. First, we consider an interface where SQL and NoSQL data can be equally tested (even though with different designs, features and goals). Second, we propose a new metric (vertices per second) that is more tailored for evaluating mobile queries over spatial big data of varying complexity. This way, we are able to measure the DBMS efficiency towards geometric attributes, instead of only measuring how fast it processes a query. Third, we provide means to compare graph-based features obtained from spatial data. Nowadays, such features are very common in large systems, but they are usually treated by two separate models (network and spatial data). Table 1.1 is a summary of the points this study tackles compared to others.

Chapter 2 describes related work and background concepts in order to fully understand spatial features. We also provide information about big data, non-relational models and DBMSs.

In Chapter 3, we present the spatial data mobile application scenario that is the base for this work. We also compare the main features of the three DBMSs considered in our evaluation. The spatial indexing features, as well as clustering and data distribution are further detailed, as they will have great impact on the performance analysis.

Chapter 4 introduces our methodology by explaining each processing stage, the dataset, the modeling characteristics and queries performed. Such discussion is important in order to demonstrate how a DBMS's performance may vary only by changing the conceptual to logical mapping applied to its dataset. Note that current DBMS benchmark methodologies focus on synthetic data generators and parameter variations to simulate real world usage – as the industry's standard TPC-H [42]. Here, the proposed workloads consider queries that reflect real operations performed by users of spatial mobile applications. We also provide alternatives to the data import tools for two of the tested DBMS.

Chapter 5 presents the experimental evaluation with setup, implementation details, parameters and results. Specifically, we evaluate three DBMSs with spatial capa-

bilities[9]: PostgreSQL[10] with PostGIS[11], MongoDB[12], and Neo4j[13] with Neo4j-Spatial[14]. Our evaluation results show that performance varies greatly in terms of algorithms and overall computational complexity due to each systems' design. Therefore, selecting a spatial DBMS heavily depends on how well it handles the targeted spatial data and the most usual functions that the system is expected to perform.

Overall, by generating a workload that mimics real data, creating and experimentally evaluating a set of queries based on mobile systems' features, this dissertation demonstrate that comparing spatial models and DBMSs requires new evaluation metrics and methodologies. The main outcome of this dissertation includes demonstrating how performance evaluation should be data and system-oriented, based on the systems' features and characteristics.

[9]Using other DBMS requires adapting the database model and queries employed in our methodology, which should be straightforward.

[10]PostgreSQL: `http://www.postgresql.org/`

[11]PostGIS: `http://postgis.net/`

[12]MongoDB: `http://www.mongodb.org/`

[13]Neo4j: `http://www.neo4j.org/`

[14]Neo4j-Spatial: `http://www.neo4j.org/develop/spatial`

Chapter 2

Background and Related Work

GIS and location-aware applications are becoming more common, sophisticated and necessary as user-based information increasingly becomes critical to all kinds of businesses. Examples include location-based advertising, customer trace data, nearby points of interest, shortest routes and recommendation systems. Governments also seek spatial computing to obtain information about events and natural disasters. As an example, Figure 2.1 depicts the area affected by hurricane Katrina in 2005.

According to Shekhar et al. [48], processing spatial objects is more complex than classical computing, as they include geometric components (points, lines, polygons and variations) with a variable number of vertices. Moreover, the volume of data processed by applications like Google Maps or Waze easily surpasses the previous generation of spatial data "heavy lifters", such as NASA's Earth Observing System and other satellite imagery programs. Those applications (Google Maps in particular) already rely on NoSQL solutions to provide their users with more spatial features and faster spatial data querying [9, 22].

Next, we explain fundamental concepts regarding both NoSQL DBMS and spatial data. We also review previous related work, show how they implemented some features and techniques regarding spatial data, big data, benchmarking, and emphasize points that were not explored in comparison with our experimental evaluation.

2.1 NoSQL

NoSQL (often interpreted as Not Only SQL) is a class of database systems that are not necessarily relational in their structure and properties. Although non relational databases have been around since the late 1960s, only recently they have conquered mass market attention [34]. Specifically, NoSQL emerged in the last decade as an al-

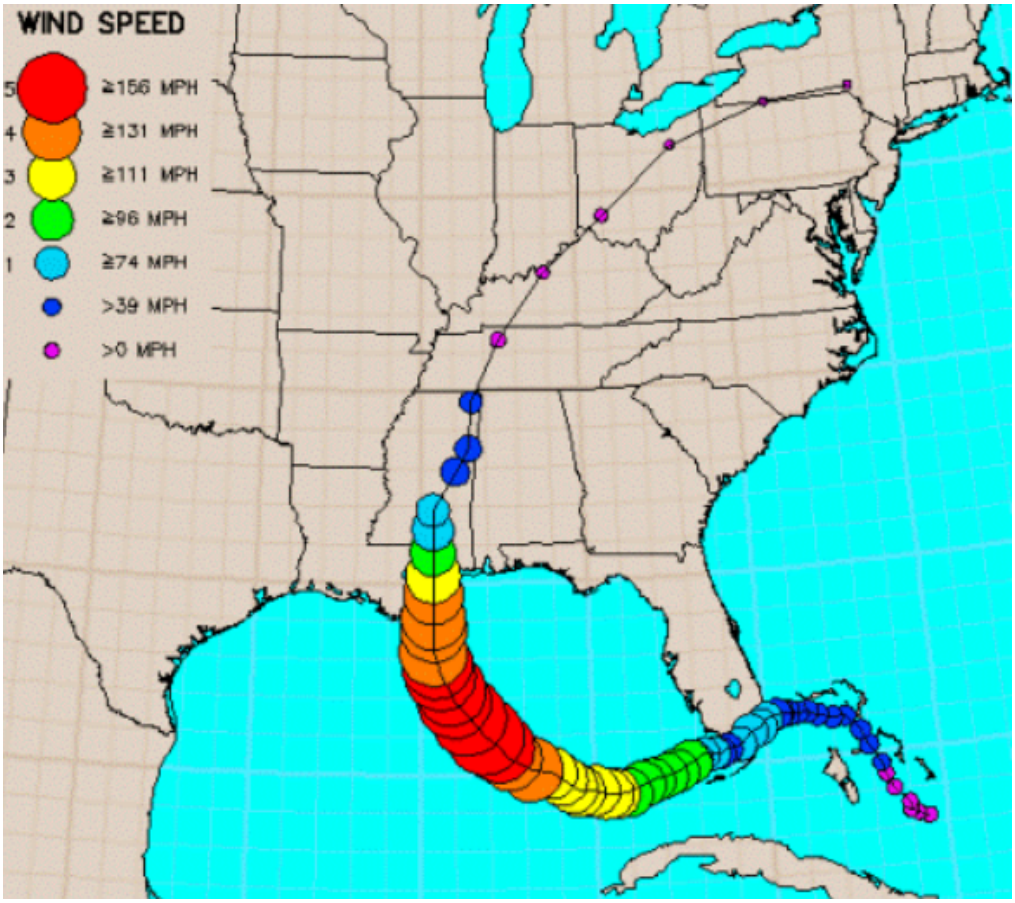


Figure 2.1: Katrina's area of effect, the size of the circle and coloration represent the hurricane's speed[1].

ternative to horizontal and scalable data growth, aiming at improving the performance of RDBMS. According to Cattel [8], the main characteristics of those systems include: horizontal scaling among machine clusters, replicability and redundancy, simple interfaces and access protocols, heterogeneous ACID (Atomicity, Consistency, Isolation, Durability) controls (while some systems maintain full ACID support, others implement only durability for example), efficient index distribution and memory usage, and dynamic attribute updates.

Over the past decade, industry has moved to a data-driven economy by requiring alternative data processing tools, mostly because RDBMS present issues when handling larger datasets. Large-scale systems have also moved from the relational environment to NoSQL [8, 18, 22, 34]. Such change increases the DBMS horizontal scalability, the ability to replicate themselves as well as index and memory efficiency. Nonetheless, NoSQL solutions also deal with weaker concurrency models by leaving some of the

[1]Jackpine Presentation ICDE 2011, Ray et al. : `http://csng.cs.toronto.edu/publication_files/195/jackpine_icde2011.pdf`

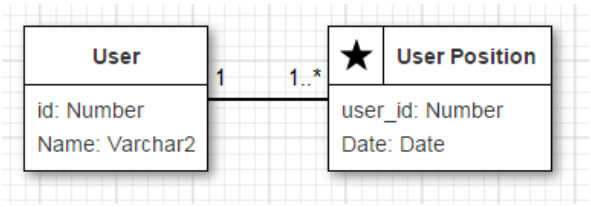


Figure 2.2: OMT-G schema fragment for user position tracking.

ACID properties for the application layer to handle [26].

Furthermore, NoSQL systems rely on already consolidated simple query languages used in other data models (such as XML[2] and JSON[3]) as a compact and efficient way of specifying their queries while maintaining compatibility among heterogeneous operational and storage systems. They also benefit from performance and usability improvements on those models and their languages [7, 20, 39, 52, 54].

Overall, NoSQL Database Management Systems are the preferred choice when performance is a priority but consistency and concurrency restrictions are not very relevant, as in applications that require lots of data retrieval with little or no updating. Such choice is even more apparent as the volume of information handled by spatial DBMSs moves towards the realm of Big Data.

Relational DBMS are challenged by two main types of NoSQL DBMS. The first is the document-oriented model (for example, MongoDB), which has already absorbed most data from the Web 2.0 applications, previously based in RDBMS [18]. The second is the Business Intelligence, Digital Libraries and Analytical Systems type, in which relational data users have moved to solutions with higher processing power, such as those based on Apache Hadoop[4].

Different NoSQL DBMS model and store data in heterogeneous ways. For example, consider an application where the user's location history must be stored in order to reduce their commute time. It searches for an alternate route, in order to avoid heavy traffic, or it suggests optimal speeds in order to avoid red traffic lights. Such an application requires the simplest association between a user class and a location class. Such data definition is represented by the OMT-G [6] model as designed in Figure 2.2. Then, in order to emphasize each model's unique features, the following figures represent the same data using other data models: Figure 2.3 for relational tables, Figure 2.4 for document-oriented model, and Figure 2.5 for a graph-oriented model. Next, we go over some of the main features of those data models.

[2]eXtensible Markup Language: `http://www.w3.org/TR/REC-xml/`
[3]JavaScript Object Notation: `http://json.org/`
[4]Apache Hadoop: `http://hadoop.apache.org/`

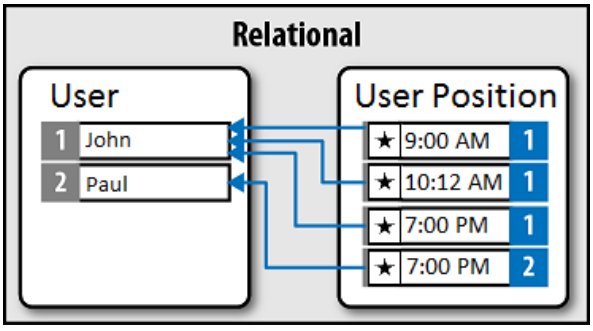


Figure 2.3: User position tracking in a relational DBMS table.

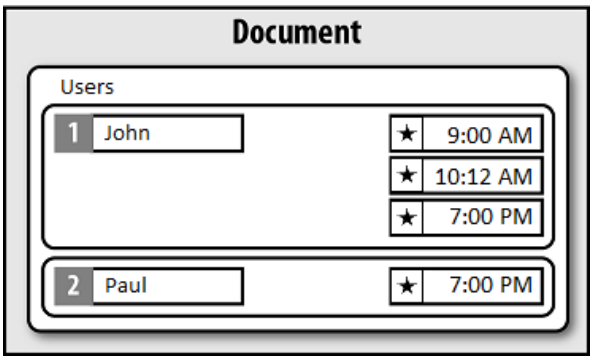


Figure 2.4: User position tracking document.

- First, Codd [12] introduced the **Relational Model**. At the time it emerged, it surpassed many other systems with the advantage of removing from the users the necessity of knowing how data is stored in the machine. It presents data as relations or collections of tables, which are based on rows and columns, and provides operators to manipulate data stored in a tabular format. It also provides a service based on actions such as queries and data modification operators, which enables the user to retrieve and store data efficiently. The relational DBMS were the most important systems since Codd's definition, turning file systems and other rudimentary database management systems obsolete.

- **Document-Oriented** DBMS are NoSQL systems designed for operating over semi-structured data usually represented as documents. As demonstrated by Banker [2], a document-based data model can represent rich, hierarchical data structures often without the multi-table joins imposed by the relational models. The data model is built for high read and write speeds. The main difference from the relational DBMS its that a document is not bound to a specific set of attributes (columns) as in the relational model. It is also more flexible by allowing designers to arrange attributes inside one another in a hierarchical manner and to add new attributes without changing the data schema. The documents

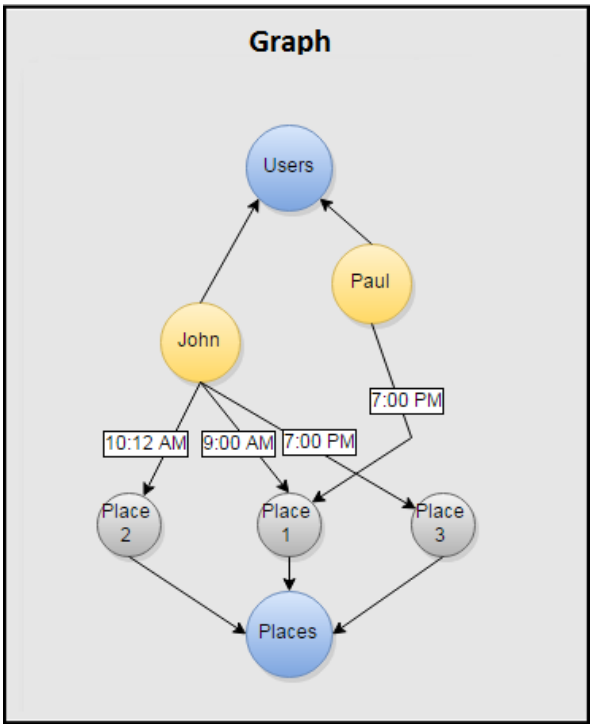


Figure 2.5: User position tracking graph.

are usually represented by a language that provides either human readability or machine attribute mapping association, or both, such as XML, JSON and YML[5].

- **Graph-Oriented** database management systems handle graph-like structures by associating data properties within nodes and relationships. The graph model enables querying a database based on a relationship attribute, which requires multiple inner joins in a traditional relational DBMS. Consequently, Graph-Oriented DBMS are usually faster for associative data, which may grow immensely larger with the popularization of online social networks. The graph-oriented database model, as pointed out by Gyssens et al. [24], allows the representation and manipulation of objects with a graph-based nature, which were previously modeled as the multi-join table sequence in the relational model.

Such a variety of choices poses challenges to system architects and their ability to stick with only one DBMS/data model. Usually one model will handle a set of queries better than the other. Finally, each model has its own challenges, as discussed in Section 4.3.

[5]YML or YAML (YAML Ain't Markup Language): `http://yaml.org/`

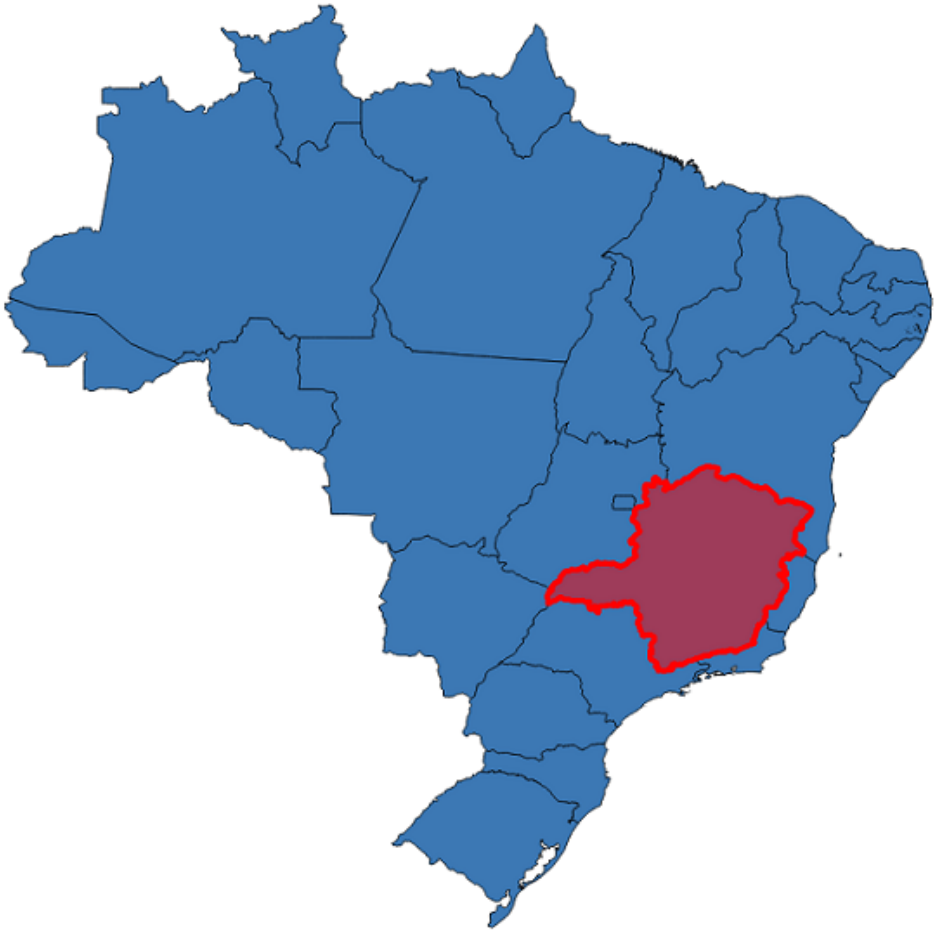
Figure 2.6: Multiple raster images of North America as a background map[6].

2.2 Spatial Data Models

There are four main models used to physically represent spatial data: raster, vector, network and spatio-temporal [47, 48]. The **raster** model uses a matrix of cells organized in rows and columns where each cell contains one or more pixels associated with an alpha-numeric value. Raster objects usually store visual, thermal and electromagnetic information obtained through aerial photographs, map scanning and satellite imagery. The information obtained is digitized and processed to provide detailed information about the targeted area. Raster data is often used as maps and map backgrounds for other GIS services. Figure 2.6 gives an example of raster data.

The **vector** model represents spatial objects as lists of vertex coordinates, thus configuring spatially-located geometric representations such as points, linestrings, polygons, multi-points, multi-linestrings and multi-polygons. Vector data represents geographic features that are related with each other over space. For example, a polygon feature may contain a point, linestring or even another polygon. Vector information is usually more compact than raster, making it easier to store in disk and accurately represent the shape and size of an object. However, it also loses other attributes such as color, temperature, etc. An example of vector data is presented in Figure 2.7, along with the list of attributes of the highlighted state of Minas Gerais.

[6]Natural Earth Raster Data: `http://www.naturalearthdata.com`



uf integer	name unknown	acronym character varying(254)	population bigint
31	Minas Gerais	MG	19595309
geometry geometry			
0103000020E6100000010000000D3100000F4E770E7A18848C0C...			

Figure 2.7: Vector data of the Brazilian states with Minas Gerais highlighted.

The **network** model is constructed when other types of spatial data (usually vector and spatio-temporal) are modeled in a graph-based representation. The network model follows the regular graph concept including vertices and edges (which are encoded as vectors and therefore are spatially located). For example, in routing applications, roads are represented in the database as edges and their intersection points as vertices, providing a traversable graph in which users search for the best route or compare possible paths in terms of both spatial (distance, number of intersections) and non-spatial (number of traffic lights, current traffic information, local maximum speed) attributes. Figures 2.8 and 2.9 illustrate vector data and its resulting graph

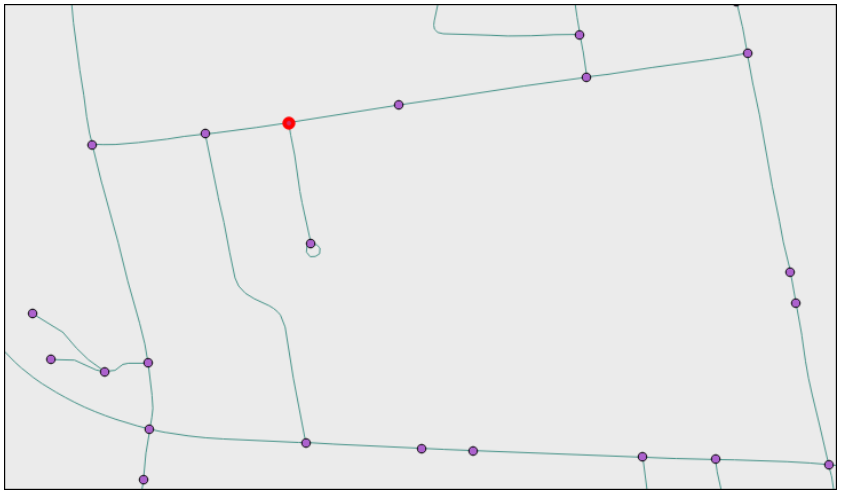


Figure 2.8: Roads (linestrings) and intersections (points) as vector data.

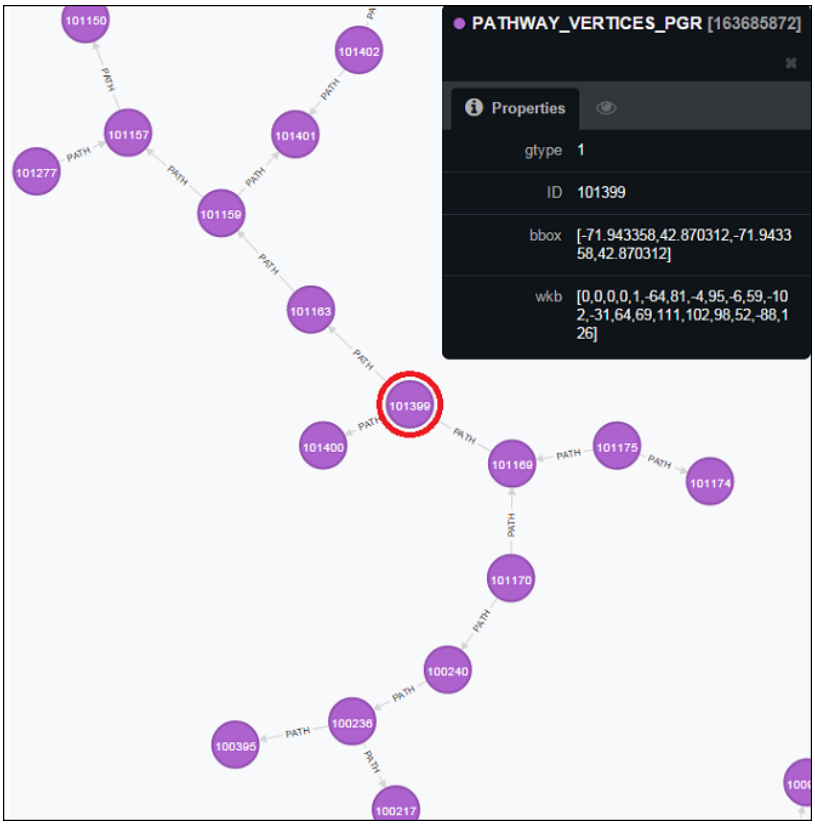


Figure 2.9: The same objects from Figure 2.8 now mapped into a traversable network.

where both represent the same location in Cheney Ave, Peterborough, NH, USA. The highlighted road intersection is also the same in both images. The node orientation in Figure 2.9 represents the node position inside the network, as opposed to its spatial orientation.

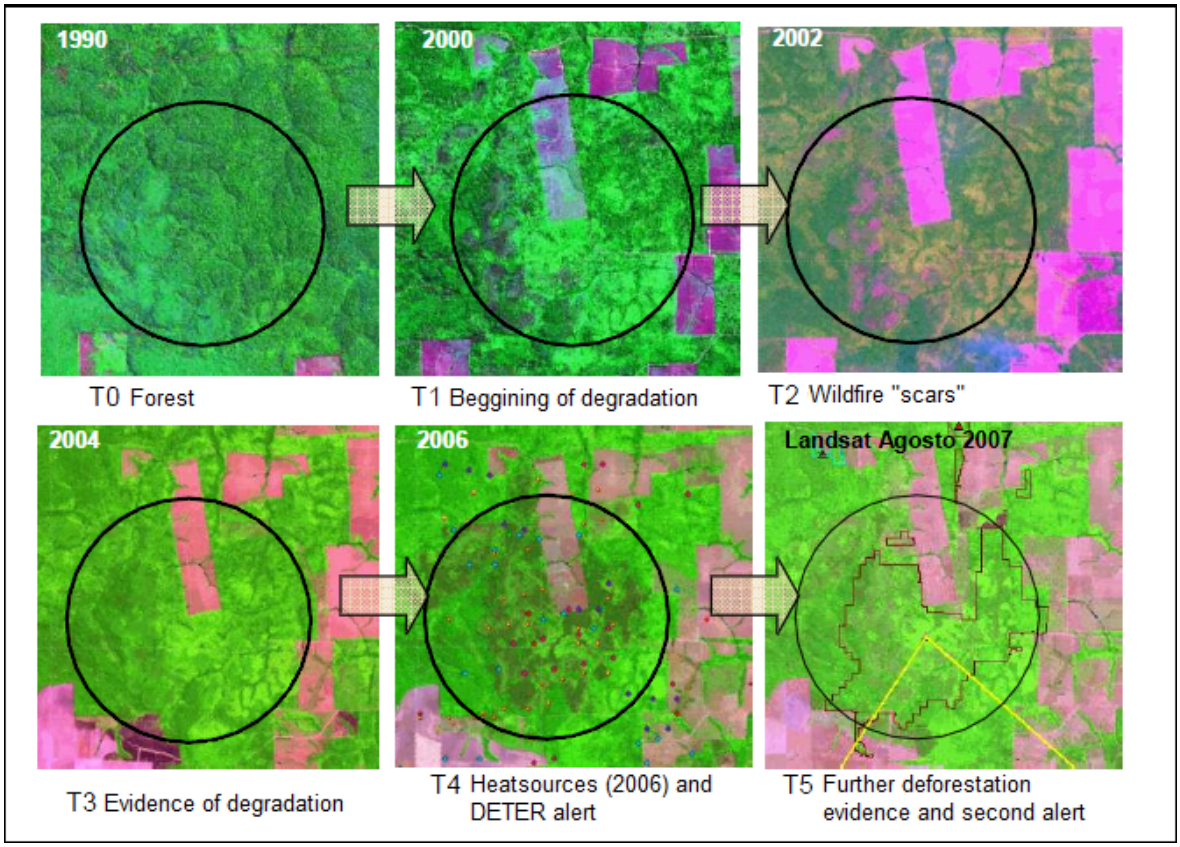


Figure 2.10: Spatio-temporal raster data monitoring the deforestation process in Itaúba, MT, Brazil, from 1990 to 2007[7].

Lastly, the **spatio-temporal** model has increasing usage in mobile devices as more information is being collected with or without the user's actual interaction. It is also present in cellphones, sensors and vehicle satellite navigation systems, as GPS trajectory applications are embedded in vehicles or installed on the drivers' cellphones [29]. Spatio-temporal data is created when one of the other models are represented in a time sequence (usually raster or vector) [22, 53]. Examples of associating with raster data is to monitor deforestation, hurricane affected areas, storms, firestorms, blizzards as well as many other phenomena. Figure 2.10 illustrates TM/Landsat raster images associated with a time series. This picture exemplifies how the deforestation process occurs, where the purple areas represent soil uncovered by vegetation and the red dots represent heat sources, probably due to man-caused forest fires.

An example of using it associated with vector data is logging a device's position throughout time in order to define an itinerary, as well as visualize possible bottlenecks and traffic jams. Figure 2.11 demonstrates spatio-temporal data with vector data. The recorded points are connected to recreate the user's trajectory throughout the day.

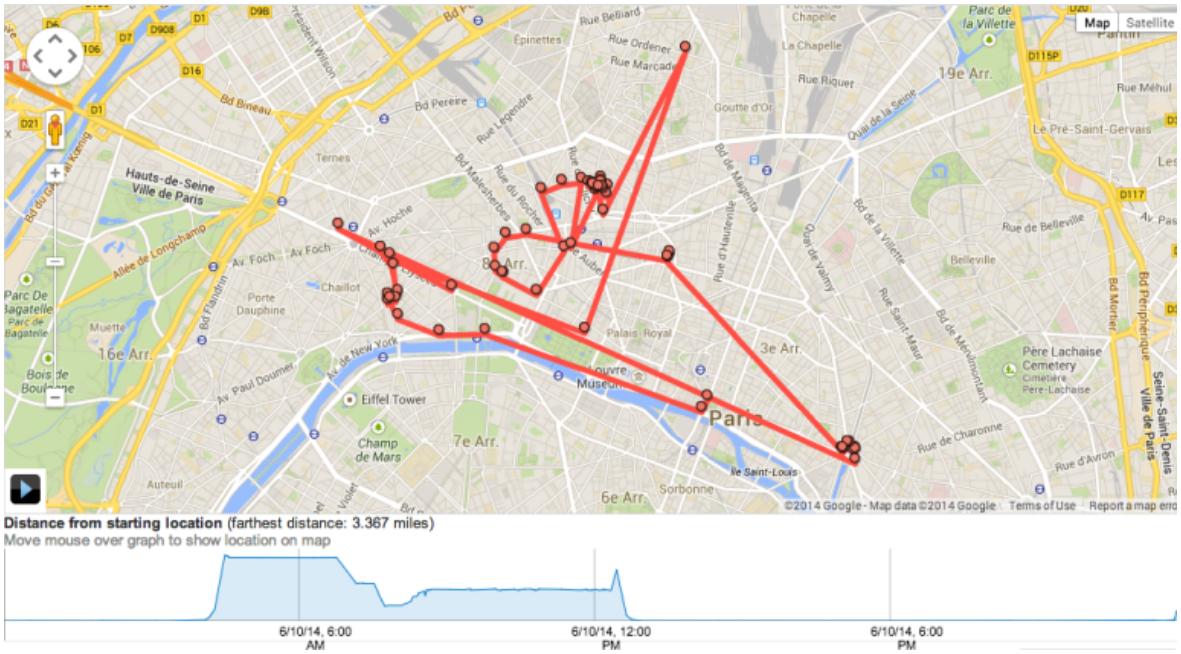


Figure 2.11: User's location history recorded by Google Maps.

2.3 Big Data

Considering the mobile scenario, it is important to discuss the concept of Big Data.

Big Data is a term that describes a dataset that has surpassed the traditional data processing systems' capabilities and present **volume**, **velocity**, **variety**, **variability** and **veracity** [30]. Such massive datasets are a challenge to both software and hardware designers, as all operations regarding Big Data require the most efficient algorithms and machinery [21, 22, 29, 32, 50]. In 2012, 2.5 exabytes of information were created on average per day on the Internet, and such number is doubling every 40 months [38]. Datasets usually grow continually because they are being fed with sensor data, software logs, mobile devices and many other types of information retrieval applications.

It is hard to deal with big data within traditional RDBMS, as it requires lots of physical space and massive parallel processing capabilities [29]. Some of the aforementioned NoSQL solutions (such as Google BigTable and Apache Hadoop) were designed specifically to tackle Big Data.

Large datasets often help in decision making, as they contain the whole data distribution and not a condensed abstract or average index, usually found in data warehouses. Big Data has the potential to provide access to information previously unknown to companies, governments, policy makers and every economy sector in a global scale [37]. Big Data analysis techniques should allow productivity growth, real-

[7]DETER system 2006/2007 scientific report: `http://www.obt.inpe.br/deter/index.html`

time detection of production failure, a faster response to natural phenomena and a more accurate reading to the general public needs.

2.4 Spatial Big Data

With the widespread use of Big Data, Shekhar et al. [48] define **Spatial Big Data** (SBD) for applications whose needs and spatial input surpass the ones of classical, relational computing. Multi-dimensional objects representing the geo-physical world are being generated by satellites and GPS-enabled devices of all kinds. From the general public to scientific and military utilization, spatial big data is one direct consequence of mobile systems' services. However, benefits obtained from big data analysis do not come cheap, as the amount of data produced and processed by those systems far exceeds the capacity of standard GIS. For example, the Moderate Resolution Imaging Spectroradiometer[8] (MODIS) satellites Terra and Aqua register up to 36 bands of the electromagnetic spectrum reflected over the Earth's surface, resulting in a huge dataset (947 megabytes every 4 days, per composite, per band[9]).

Shekhar et al [48] also point out that Spatial Big Data will require the development of new database management technologies, but with such advancements, it will also provide more research and industry opportunities. Examples include the quicker detection of disease outbreaks, the effects of geographic attributes in social networks, large scale collaboration studies, among others. It will also bring more responsibility, as governments and companies with access to such massive databases will be able to monitor everything and everyone, anywhere and everywhere on the planet, in real-time.

2.5 Discussion on Related Work

Here we discuss how different studies have contributed to the Spatial DBMS scenario, and how they have left room for improvement. With the advent of NoSQL DBMS, spatial data has gained lots of ground and entered the Big Data era without much progress in both performance and features provided by their old RDBMS counterparts.

On spatial data heterogeneity, Baptista et al. [3] showed how to interoperate spatial data in SQL and NoSQL databases by using OGC[10] services. They also discuss the importance of handling spatially-enabled social networks data. This study brings up the importance of different storage techniques in spatial web services and how the

[8]MODIS: `http://modis.gsfc.nasa.gov/about/`

[9]Statistics Canada Crop Condition Assessment Program: `http://www.statcan.gc.ca`

[10]Open Geospatial Consortium: `http://www.opengeospatial.org/`

NoSQL DBMS have incorporated spatial features to handle spatially-enabled social networks data.

Likewise, Hora et al. [28] have presented a method for mapping network relationships from spatial databases to the Geography Markup Language (GML) – an XML extension for spatial data. Note that there are limitations regarding GeoJSON (a JSON extension for handling spatial data), and their approach could point to a solution in such front as well. Extending [28] could also potentially provide spatial networking (urban routing, water, sewage and electric networks) features to any DBMS that uses JSON, such as MongoDB, for example.

Regarding performance evaluation of spatial databases, recent studies on spatial joins discuss how applications rely on fast spatial processing methods [49, 51]. They also show that there is still much room for improvement regarding spatial queries: performance gains are noticeable by changing the indexes' data structure and the access algorithms, as well as using multi-threaded implementations. Specifically, for the mobile scenario, there are many particularities regarding data traffic, speed variations under different areas of coverage and mobile devices. Those characteristics also influence the GIS design, as immediate, limitless data requests may not be very efficient carrier-wise. Liu et al. [35] propose a method that compensates the lack of features that many of those applications suffer when going offline by improving their accessibility with a probabilistic query window prediction. The algorithm fetches in advance data that may be requested by the user, maintaining a smoother user experience while disconnected from the network.

Still regarding performance, benchmarking techniques have been around since the first performance issues became essential when choosing a DBMS. As for spatial data benchmarking, the Spatial Star Schema Benchmark [40] has a synthetic data generator specific for spatial data warehouses. Its spatial geometry generator uses a recursive partitioning of an initial envelope, dividing it in quadrants. The partition produces a quadtree of minimum bounding rectangles that are filled with a random geometric shape.

For big data, the benchmarking tool proposed by Ghazal et al. [19] also includes a synthetic data generator and a data model. Likewise, Chen [10] proposed a functional workload model for big data. It tries to cover some of the many problems when benchmarking big data, as there are no standard benchmarks currently available. They propose several business cases aiming at many industry standard scenarios already covered by previous benchmarks such as the Transaction Processing Performance Council's TPC-C. The TPC-C benchmark is a tool for comparing online transaction processing (OLTP) performance on various hardware and software configurations. By extend-

ing TPC-C and other benchmark combined features, Ghazal et al. propose several big data benchmark characteristics, taking the TPC-C benchmark as the yardstick. They also provide many concepts like functions of abstraction, models and map-reduce translations of those concepts.

Even though synthetic datasets are usually favored for evaluating queries over string and numeric attributes, the equivalent synthetic generated geometries are not similar to real-world data. Specifically, the randomness of the generator results in overly squared polygons and unnatural land shapes, as demonstrated by Baptista et al. [3]. Furthermore, the mix of man-made and natural features usually found in GIS varies according to the region and is strongly related to the application, making it harder to accurately represent real workloads [47]. Therefore, in this work we use real datasets, as described later.

On the relational world, Jackpine is a more generic benchmark for spatial relational databases [43]. It tests topological predicates and spatial analysis functions in isolation, as well as six typical spatial data application scenarios. Its dataset comes from the state of Texas, including county and municipal divisions, hydrographic and vegetation data, roads and highways. Nonetheless Jackpine is not ideal for evaluating spatial big data for mobile applications. First, it covers only relational databases, and spatial Big Data applications usually employ NoSQL as well. Second, it is not appropriate for measuring the effects of data volume and complexity (e.g., spatially joining regions delimited by polygons with thousands of vertices to sets of millions of points). Third, mobile applications often need network-based attributes and functions, which may be cumbersome (if not impossible) to implement over a relational system.

Now, considering the distributed processing of spatial data, database clusters for relational queries face an architectural challenge on improving performance and using all the available hardware as computer clusters became cheaper, more common and readily available [1]. Operations that nowadays are easily handled in parallel by NoSQL DBMS, are much harder to coordinate and distribute over RDBMS clusters. The design scheme must consider many usage and workload variables, which lead to mixed physical designs and different levels of parallelism, depending on the query/modification performed.

Likewise, Le Pape et al. [33] proposed a technique to improve query performance in database clusters through freshness control. The freshness of an object is calculated based on the frequency in which the object is accessed and how long it has been since that object was requested by a query. They also demonstrated how relational DBMS had to be extensively modified in order to extract optimal performance over clusters. Their query routing algorithm managed to improve the cluster's throughput but it still

does not address many big data requirements, such as constant change of the data group being read (which goes against the freshness strategy).

Röhm [44] combined a query routing algorithm with a physical design arrangement in order to increase performance over a database cluster. Such effort over a RDBMS greatly improves its performance. The two stages could be more easily implemented over NoSQL databases, as the querying algorithm and physical design can be improved separately. Such improvement is possible as NoSQL DBMS provide a native clustering interface, leading to an experience that is similar, if not equal, to a single-machine system.

Cluster and cloud computing are very powerful tools when dealing with large amounts of data. Methods to evaluate DBMSs over such structures would have to monitor many more variables, and it would require a much more controlled environment in order to perform a fair comparison. Therefore, this work does *not* consider such multi-machine scenario.

Our evaluation, as explained previously, differs from the previous work by considering SQL and NoSQL DBMSs, proposing a new performance metric (vertices per second) and providing ways to compare graph-based features obtained from spatial data.

Chapter 3

Spatial DBMS

Spatial data extensions already appear in several DBMS. However, the programming languages with which they are built, their design patterns and architectural characteristics can weigh upon their efficiency. Also when focusing on the mobile systems scenario, data properties and volume vary greatly from other GIS. This chapter follows with more information about the mobile systems' features, specific architecture information and how spatial indexing is performed among the tested DBMS.

3.1 Mobile Systems

Mobile systems work over the Internet (and its communications links) through computing devices that are around in the field. Mobile computing applications can be roughly classified into location-based services, sensor networks, and ad hoc networking. Their technical challenges include interaction between anonymous entities, timely decoupled data processing, and potentially millions of mobile clients and devices. According to Cugola and Jacobsen [13], a simple taxonomy for mobile systems classifies the nature of the device mobility and the nature of the network infrastructure deployed as: (*a.1*) the computing devices are stationary while online, but join and leave the network at different, physically distributed access points; (*a.2*) or the devices also move during connection; (*b.1*) a fixed network infrastructure is deployed; (*b.2*) or no external network infrastructure is available. The combinations of these two dimensions define different types of mobile applications. Nonetheless, the common point is: mobile applications, services and systems *collect* and *provide* information to their users at the same time.

Then, the technical challenges vary according to the type of mobile device and software. Take for example Location-based Services (LBS) [45]. LBS are defined as

applications that integrate geographic location (i.e., spatial coordinates or position) with other information in order to provide valuable services to their users. Examples include car navigation systems, tourism websites, location-aware recommendation systems, points-of-interest review systems (e.g., restaurants and stores), traffic information services, fleet management, car and assets tracking, gaming, local advertisement, among many others. All such services have become very popular with the development of mobile communications and the ever growing access of Internet connected devices (cars, cell phones, PDAs, tablets, etc).

The amount of information flowing on general mobile systems (and LBS) is huge. The main reason is because such systems receive the user's location and provide both *push* and *pull* services. For example, consider the scenario of m-commerce (*mobile* commerce) and advertising. Given a user's location, a push service may send a discount voucher from a nearby store. Likewise, in a pull service, the user may request cheap restaurants in the local area. With such big volume of information (big data) come many problems from not only the data per se (collecting, treating, filtering, storing, transmitting) but also privacy, security, time performance, communication, network infrastructure, availability, and so on [45, 48].

Such constant traffic and processing of spatial data require faster server responses. As the number of devices (smartphones, in-vehicle navigation devices, etc) rapidly increases, mobile applications have become one of the biggest spatial data providers [41]. Indeed, even systems and applications that do not *directly* deal with spatial data usually employ spatial metadata attributes. For example, when taking a picture on a smartphone, geographical metadata is stored within the picture's file, enabling to trace the photo back to the location it was taken, as illustrated in Figure 3.1, whose data is in Table 3.1. Other examples include instant messaging applications, which trace the user's location and enable the other party to view it on a map.

In this dissertation, we take the data management perspective and consider the requirements of the broad category of mobile systems, which employ user and devices' location as a key data. Specifically, our focus is on the challenges posed by different data models handling the same dataset, such as: managing millions of data insertion and queries; managing a (potentially) very large number of information providers (not only the mobile devices but the service and application data producers); and process diverse content formats, ranging from pure text documents to complex geometry polygons. To deal with such challenges we trust on SQL and NoSQL DBMS with geographic features, as explained in the next sections.

Figure 3.1: Outskirts of Hong Kong.

Table 3.1: Spatial metadata obtained from Figure 3.1.

Attribute key	**Value**
GPS Version ID	2.2.0.0
GPS Latitude	22.413932 degrees
GPS Longitude	114.271813 degrees
GPS Altitude Ref	Above Sea Level
GPS Altitude	65 m
GPS Satellites	05
Modify Date	2010:03:04 01:05:15
Create Date	2010:03:03 16:57:47

3.2 DBMS Main Features

In this dissertation, we consider one DBMS representing each of the three models analyzed (relational, document-oriented and graph-based). All three provide spatial extensions and a license-free installation[1]. They also represent different types of file management and query processing and optimization: PostgreSQL version 9.3.4 x64 with PostGIS 2.1.3, MongoDB 2.6.3, and Neo4j Community 2.1.4 with Neo4j Spatial 0.13. We now provide a brief description of each DBMS, and discuss their indexing, clustering and data distribution. Table 3.2 shows a brief comparison[2] between the DBMS analyzed in this study.

The first DBMS considered in our study is the relational DBMS **PostgreSQL**

[1]Other systems may fulfill these criteria. However, we opted for choosing three representatives based on DB-Engines ranks such as: DBMS and database model popularities.

[2]DB-Engines: `http://db-engines.com/`

Table 3.2: DBMS Comparison.

	PostgreSQL	**MongoDB**	**Neo4j**
Website	www.postgresql.org	www.mongodb.org	neo4j.com
First release	1995	2009	2007
License	Open Source (BSD)	Open Source (AGPL 3)	Open Source (GPL 3)
Implemented language	C	C++	Java
Database model	Relational DBMS	Document store	Graph DBMS
Data scheme	yes	schema-free	schema-free
Typing	yes	yes	yes
Secondary indexes	yes	yes	yes
SQL	yes	no	no
APIs	native C library, streaming API for large objects, JDBC, ODBC	proprietary protocol using JSON	Cypher query language, Java API, RESTful HTTP API
Server-side scripts	user defined functions	JavaScript	yes
Triggers	yes	no	yes
Partitioning methods	no, but can be realized using table inheritance	Sharding	Enterprise Edition only
Replication methods	Master-slave	Master-slave	Master-slave
MapReduce	no	yes	no
Consistency concepts	Immediate Consistency	Eventual Consistency and Immediate Consistency	Eventual Consistency
Foreign keys	yes	no	yes
Transaction concepts	ACID	no	ACID
Concurrency	yes	yes	yes
Durability	yes	yes	yes
User concepts	fine grained access rights according to SQL-standard	Access rights for users and roles	no

with PostGIS. PostgreSQL is an open source object-relational system first released in 1995. PostgreSQL supports all major operating systems, is ACID and ANSI-SQL:2008 compliant. PostGIS is an open source geographic extension for PostgreSQL that allows spatial queries to run on SQL. It was first released in 2001 and supports geometry types for points, linestrings, polygons, multipoints, multilinestrings, multipolygons and geometry collections. PostGIS supports an R-tree-over-GiST (Generalized Search Tree) spatial index and implements most spatial predicates and operators defined by the OGC.

The second DBMS is the NoSQL document-oriented **MongoDB.** MongoDB is an open source document DBMS that aims to provide high performance, availability and horizontal scalability. Its data model is based on the key-value association by

using JavaScript Object Notation (JSON). It provides a query mechanism called *query by example*, where the query is an object to be compared with existing database elements, and all matching objects are returned. MongoDB's features include flexible data collections, sharded clusters, automatic sharding and replicated datasets. MongoDB spatial features are based on the GeoJSON format, a JSON's extension that specifies geometry objects such as points, linestrings, polygons, multipoints, multilinestrings and multipolygons. GeoJSON queries on MongoDB require an existing spatial index, and the spatial queries have the following operators available: inclusion, intersection and proximity.

The third DBMS is the NoSQL graph database **Neo4j with Neo4j-Spatial.** Neo4j is a graph-oriented open source DBMS. Its core features include data model flexibility based on graph elements such as nodes, edges and attributes, graph traversing performance and scalability, supporting billions of nodes and relationships and full ACID transactions. Neo4j Spatial is an extension that provides spatial querying over geometry attributes stored in nodes. Furthermore, its spatial indexes are built over the graph structure provided by Neo4j. It also implements most OGC spatial predicates and operators.

3.3 Spatial Indexing

Spatial indexes (as any regular database index) provide faster ways to access spatial data by avoiding sequential reads and usually traversing a more efficient search structure. Regular database indexes usually rely on decomposing and serializing alphanumerical values in order to associate them with an index structure (a B-tree, a hash table and others). Such approaches are not ideal when paired with spatial objects, as these objects possess multidimensional attributes (geometries) that cannot be directly added to a B-tree, for example. The solution often is some level of data serialization/hashing to actually fit the objects, as opposed to indexing data structures specifically designed to handle multidimensional data (R-tree, Quadtree, etc.).

Here we describe the two strategies used by the evaluated systems, the R-tree [23] and the combination of Geohash and a space-filling curve. The **R-tree** is a data structure used for optimizing spatial data access. It indexes multi-dimensional information such as coordinates, polygons and other geographical attributes through minimum bounding rectangles of spatial objects modeled as tree entries, where each entry is inserted based on the position of each rectangle. The R-tree operates similarly to the B-tree, as both are balanced search trees organized in pages, with each page contain-

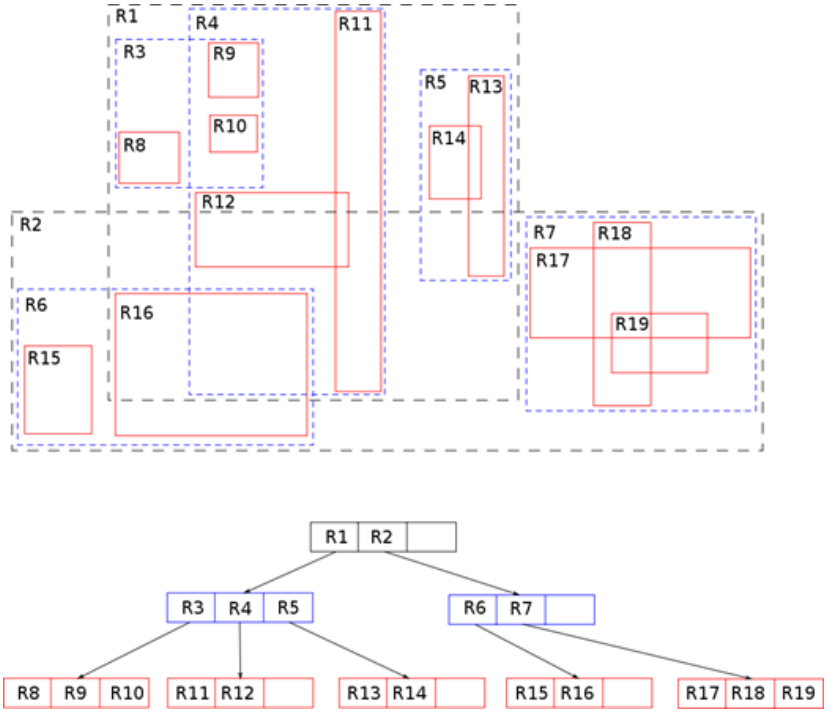


Figure 3.2: R-tree example, minimum bounding rectangles over space and the respective tree structure.

ing a maximum number of entries. The R-tree does not ensure a good worst-case performance, therefore variants and extensions were proposed in order to tackle such deficiencies, for example the R* tree, R+ tree, the Hilbert R-tree and others [5, 46, 31]. Figure 3.2 illustrates an example of R-tree with the minimum bounding boxes at the top and the respective tree at the bottom.

The second strategy is a combination between Geohash and a space-filling curve. The **Geohash** is a code system based on latitude and longitude. It is a hierarchical spatial data structure that subdivides the space into grids. Each grid represents a square/rectangular section of the globe where a prefix is associated with that area. The closer the prefix numbers are to each other, the closer the geographic grids are from one another. The **space-filling** curves are curves that contain an entire multi-dimensional range of values over every point of unit square. Hence, when associated with Geohash, a curve contains every spatial grid generated by Geohash mapped into a single-dimension attribute.

Each DBMS in this study includes different strategies and implementations of spatial indexes. First, PostGIS uses the GiST [27] R-trees for spatial attributes. The GiST R-tree is a generalized implementation of the traditional R-tree, using PostgreSQL's extensibility features and adapted to PostGIS's spatial representation types. The main difference between a GiST R-tree and the traditional R-tree is that GiST

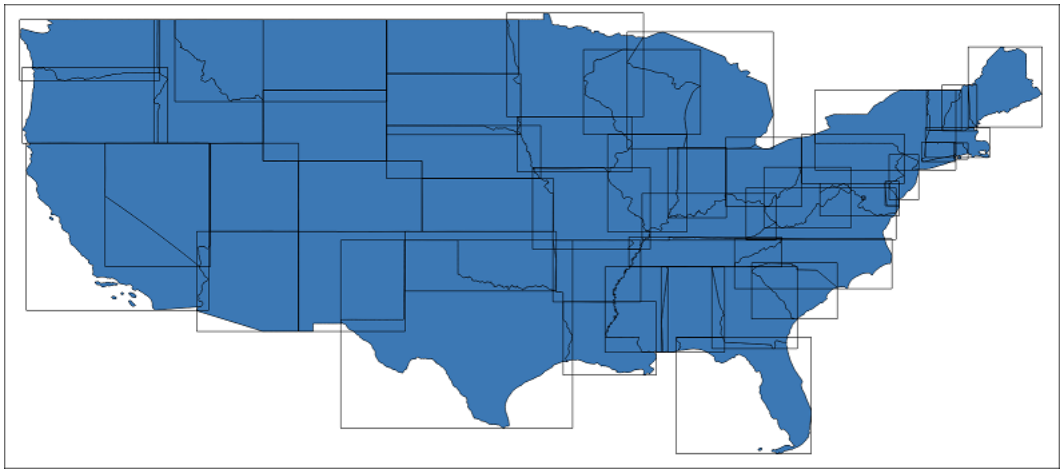


Figure 3.3: Continental USA states' bounding boxes.

indexes are lossy, as each document is represented in the index by a fixed-length signature. In other words, each hit on the index may be a false positive, so the filtered rows must be retrieved and evaluated to determine if their values match the query or not. Depending on the data stored, the GiST R-tree may be slower than the R-tree by a small factor, but it is frequently adopted because it is a common interface where multiple index types are built on. Figure 3.3 illustrates the bounding boxes that compose the R-Tree index over the continental states of the USA.

Second, Neo4j Spatial uses an R-tree built over its graph structure, where the tree root is a new node in the graph. Relationships connect the root to the nodes with spatial attributes, adding them to an R-tree that uses the graph structure, called *spatial layer*. Neo4j Spatial does not allow querying nodes spatially without adding them to the spatial index/layer first. Apart from the graph-based structure, Neo4j's R-tree behaves as a regular R-tree.

Third, MongoDB uses a combination of a space-filling curve with Geohash, as illustrated in Figure 3.4, then adding its values into a B-tree. As its main indexing tool relies on a B-tree, its spatial attributes modeled as GeoJSON inherit the index structure. However, since spatial data is multidimensional, it requires dealing with the dimensionality problem. The solution is to adopt a space-filling curve, a continuous line in a two-dimensional plane that intersects every point (MongoDB does not natively support geometries with three or more dimensions). After the space-filling curve, it applies geohashing to the coordinate values and inserts them in the B-tree. This approach is not optimal when dealing with spatial data as serialization of multidimensional objects degrades the performance of many spatial queries, such as looking for the nearest neighbor. When two neighbors are spatially close, they may be widely

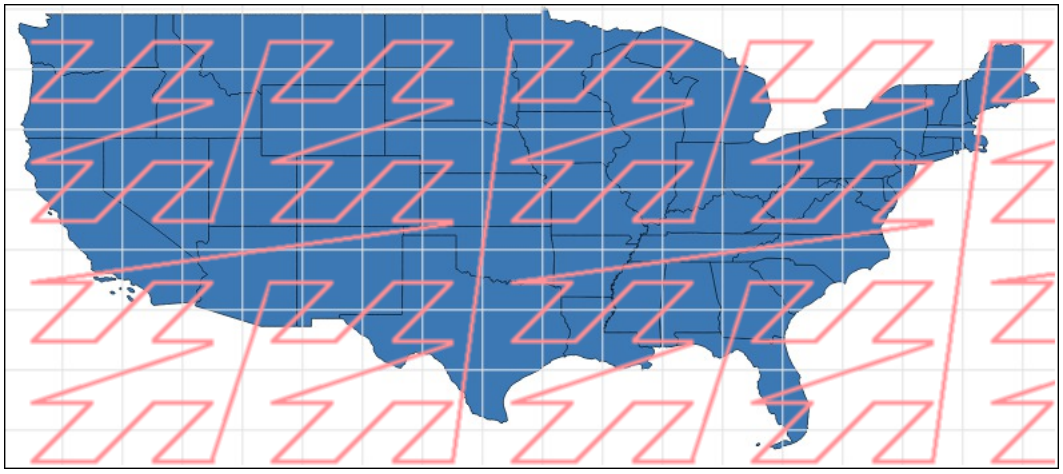


Figure 3.4: Continental USA states' Geohash space-filling curve combination.

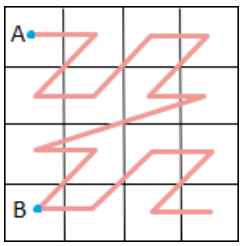


Figure 3.5: Z curve distance example.

separated when looking at the space-filling curve. This phenomenon is exemplified in Figure 3.5 where point A is only two tiles apart from point B, but in the curve, they are nine tile distances apart (in terms of id sequence, not the actual distance).

Chapter 4

Comparison Methodology

Now, we present our methodology for comparing spatially-enabled DBMSs considering four main requisites. The **first** one is to use real-world spatial data. We believe that as spatial queries and indexes are usually more complex and computationally expensive, it is imperative to consider data as close to reality as possible. The **second** goal is to define a set of queries, insertions and updates that focus on mobile systems, as the mobile applications scenario potentially uses and produces large quantities of spatial information. The **third** goal is to create a methodology that offers implementable interfaces where different spatial DBMSs may be easily added to the performance evaluation. Such goal maintains compatibility with new software versions and database management systems by making it easier to perform extended comparative analysis in the future. The **last** is to demonstrate how performance evaluation for GIS must be data and feature-oriented, in opposition to the current generic benchmark tools that have wide coverage but little specificity. By comparing systems derived from different models, strategies and implementations, we are able to compare each systems' performances and analyze which model accommodates best the desired application. Next, we describe our methodology's processing stages (Section 4.1), the dataset used as a workload (Section 4.2), the modeling and mapping strategies adopted (Section 4.3), the data loading procedure (Section 4.4), improvements implemented to speed up the loading process (Section 4.5) and the proposed queries (Section 4.6).

4.1 Processing Stages

The comparison methodology follows five stages: data loading, workload generating, warm-up, evaluation and shutdown. The data loading, workload generating and evaluation stages should be run separately from the other ones. Running any step should

Table 4.1: DBMS Warm Up time.

DBMS	Warm up time
PostgreSQL	0.083 hour
MongoDB	0.1 hour
Neo4j	2.33 hours

also trigger the shutdown stage automatically at the end of the run and any resources allocated by the systems should be freed. Even though we measured the time spent in both data loading and warm up stages, they are not accounted during the final performance evaluation, as such stages are performed in much less regular basis than querying, modifying and inserting data.

Data Loading. This stage loads shapefiles[1] (a popular geospatial data format) into the databases that are going to be analyzed. This step runs separately, as spatial big data loading may take several hours or even days, depending on the DBMS's importing tools.

Workload Generation. The second stage generates the workload to be processed. The options are: creating data tables and structures by using the input shapefiles directly; or using a DBMS with the dataset previously loaded (at data loading). Here, the necessary parameters (to be discussed later as shown in Table 5.2) for the evaluation stage are also extracted from the same shapefile spatial dataset. This procedure defines queries with the least amount of null results or sequential scans, making sure that the DBMS uses the indexing and cache mechanisms available. The result from queries that select random entries in the shapefiles or the DBMS with loaded data is then written in a file to be read by the evaluation stage.

Warm-Up. This phase performs several queries to load indexes and other memory-related attributes that affect the DBMS's performance. When it ends, the amount of memory consumed by the DBMS processes and its memory mappers (if that is the case) are recorded. The warm-up phase is necessary as DBMSs in general try to avoid disk requests by storing index references in RAM, which is also how queries will be processed in production. The average warm up time for each DBMS is presented in Table 4.1.

Evaluation. Now, pre-defined queries are executed over each DBMS individually by using the parameters obtained in the second phase. Our methodology aims to evaluate and compare the performance of DBMS to be used by mobile systems and applications. Therefore, groups of queries were created based on such features, which include map

[1]ESRI Shapefile: `http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf`

browsing, zooming and panning, proximity searching, and urban routing (finding the shortest path between two points).

Shutdown. The shutdown phase is where each DBMS's connections are shutdown to ensure data integrity, finalizing any pending transactions and releasing read and write locks, making sure that the database is in a consistent state. It is also important to verify if any system resources consumed by the DBMS during that session were not freed or returned to the operating system, which may point to memory leaks and other memory management issues. The methodology also has a shutdown hook, which means that even if the evaluation process is manually canceled or fails, the shutdown procedure will still be performed by a separate thread, thus guaranteeing data integrity and that no open resources remain connected to the database systems.

4.2 Dataset

Spatial datasets are increasing in size and coverage, but only a few of them may be classified as big data. In this study, we opted for using real-world data, with realistic spatial distribution (i.e., a varying density of objects through space, from crowded downtown areas to rural spaces), covering a large territory and including a variety of representation types. Also, there should be a mix of natural (rivers, lakes, plateaus, continents, islands, etc) and man-made objects (streets, highways, blocks, cities, states, countries, etc) in the dataset, so that the number of vertices per object and the object distribution along space varies widely. The TIGER/Line (Topologically Integrated Geographic Encoding and Referencing[2]) dataset is public and covers all aforementioned features.

The 2013 TIGER/Line shapefile set provided by the U.S. Census Bureau is one of the most complete and well documented datasets regarding the USA land and overseas territory. The USA is one of the countries with the largest number of mobile devices connected to the Internet[3] and consequently one with the most people using spatial-aware systems, as mentioned in Section 1. Initially a simple street centerline database, TIGER/Line has evolved to include a number of additional features that support Census activities in the USA. Running specific queries on the 2013 TIGER dataset generates a workload that serves as a close-to-reality evaluator when comparing different DBMS and analyzing how they respond to queries over spatial big data.

[2]TIGER/Line: `https://www.census.gov/geo/maps-data/data/tiger-line.html`

[3]Cisco Visual Networking Index: `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html`

Figure 4.1: 2013 TIGER/Line collections AREALM, AREAWATER, ROADS, PRISECROADS, PRIMARYROADS and POINTLM at the Manhattan's Central Park.

The TIGER shapefiles contain vector geospatial data. Both shapefiles (.shp) and database files (.dbf) are separated into census blocks by a hierarchical framework that groups files depending on their spatial coverage, such as nation, state and county-based files. The shapefiles include polygon boundaries of geographic areas, linear features (roads, hydrography and others) and point features. They do not contain any sensitive data, areas used for administering censuses, military information, surveys and other internal processing attributes. Figure 4.1 shows a portion of the vector dataset in the QGIS[4] map viewer. The dataset classes are presented in Table 4.2.

[4]QGIS 2.5: `http://www.qgis.org/en/site/`

Table 4.2: 2013 TIGER/Line vector collections.

Folder Name	Shapefile/Relationship File
ADDR	Address Range Relationship File
ADDRFEAT	Address Range Feature
ADDRFN	Address Range-Feature Name Relationship File
AIANNH	American Indian/Alaska Native/Native Hawaiian Areas
AITSN	American Indian Tribal Subdivision National
ANRC	Alaska Native Regional Corporation
AREALM	Area Landmark
AREAWATER	Area Hydrography
BG	Block Group
CBSA	Metropolitan Statistical Area / Micropolitan Statistical Area
CD	Congressional District
CNECTA	Combined New England City and Town Area
COASTLINE	Coastline
CONCITY	Consolidated City
COUNTY	County
COUSUB	County Subdivision
CSA	Combined Statistical Area
EDGES	All Lines
ELSD	Elementary School District
ESTATE	Estates
FACES	Topological Faces (Polygons With All Geocodes)
FACESAH	Topological Faces-Area Hydrography Relationship File
FACESAL	Topological Faces-Area Landmark Relationship File
FACESMIL	Topological Faces-Military Installation Relationship File
FEATNAMES	Feature Names Relationship File
LINEARWATER	Linear Hydrography
METDIV	Metropolitan Division
MIL	Military Installation
NECTA	New England City and Town Area
NECTADIV	New England City and Town Area Division
OTHERID	Other Identifiers
PLACE	Place
POINTLM	Point Landmark
PRIMARYROADS	Primary Roads
PRISECROADS	Primary and Secondary Roads
PUMA	Public Use Microdata Area
RAILS	Rails
ROADS	All Roads
SCSD	Secondary School District
SLDL	State Legislative District –Lower Chamber
SLDU	State Legislative District –Upper Chamber
STATE	State and Equivalent
SUBBARRIO	SubMinor Civil Division (Subbarrios in Puerto Rico)
TABBLOCK	Tabulation (Census) Block
TBG	Tribal Block Group
TRACT	Census Tract
TTRACT	Tribal Census Tract
UAC	Urban Area/Urban Cluster
UNSD	Unified School District
ZCTA5	5-Digit ZIP Code Tabulation Area

4.3 Data Modeling

Many spatial data modeling methods extend the entity-relationship model for describing geographic data. While the result of such data model extensions is usually associated with relational database management systems, other non-relational spatial solutions are being used by very large GIS. With the NoSQL arrays taking over Big Data, data models that easily fit relational DBMS are being translated and adapted into other systems. However, this may not prove optimal when handling large amounts of data, as it is not so clear how to define which mapping strategy (or even more than one) to maintain, either on relational or non-relational DBMS. Modeling strategies based on the system requirements and features are necessary. Nonetheless, it is also important to evaluate how beneficial the adoption of multiple data mappings of the same dataset can be.

When modeling spatial attributes, one must consider that the association of primary and foreign keys applied to numbers and strings provides an easy way to join tables, which is not possible with geospatial attributes. Spatial data is a multi-dimensional attribute that by its nature is not a static join attribute, as the interaction between two spatial objects may generate more than two outcomes (some of which are not mutually exclusive) in opposition to the binary outcome of a numerical foreign key to the targeted numerical primary key. The outcome of such spatial relationships must be calculated dynamically as demonstrated next.

The model proposed by Clementini and Egenhofer [11, 17] became the standard procedure to associate two geospatial objects. It translates the relationship between two geometries into a set of outcomes based on a decision tree. An example of the outcome matrix, called the Dimensionally Extended nine-Intersection Model (DE-9IM) can be seen in Figure 4.2. Note that the intersection of the interiors is a two-dimensional area, so that matrix cell's value equals 2, indicating that the object resulting from that operation is a two-dimension object, in other words, a polygon. When intersections are over single lines, that matrix cell's value equals 1, indicating that a one dimensional object is the result. When the polygons touch over single points, that portion of the matrix equals 0, which indicates the 0-dimensional point objects as results. When there is no intersection between components, the respective matrix value is set to a Boolean false.

[5]OpenGeo Suite: `http://suite.opengeo.org/`

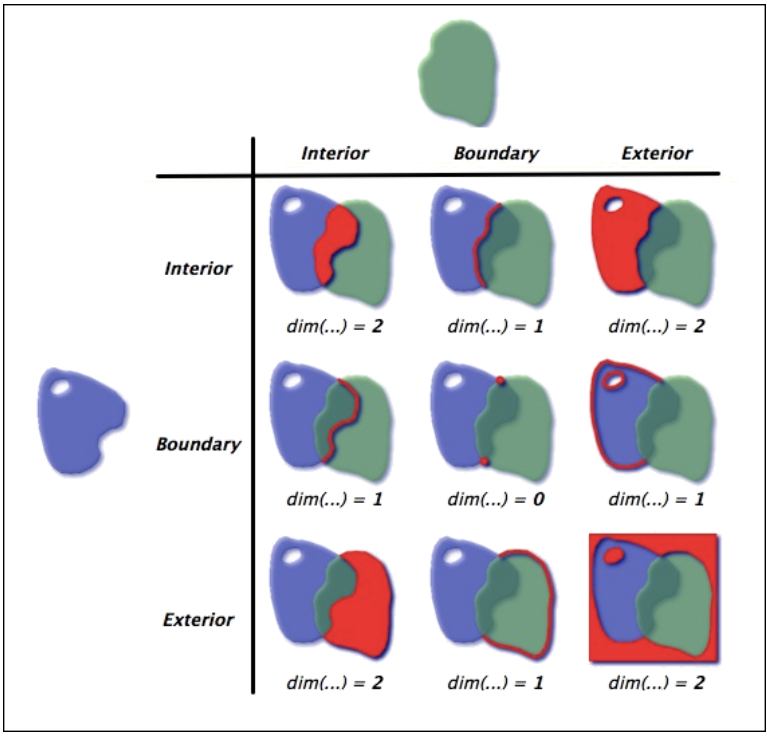


Figure 4.2: DE-9IM over spatial object interactions[5].

4.3.1 OMT-G

Considering that the spatial relations cannot be represented as foreign key associations, Borges et al. [6] demonstrated how spatial data characteristics make modeling more complex and proposed the *Object Modeling Technique for Geographic Applications* (OMT-G). The OMT-G provides conceptual, presentation and implementation abstraction levels and it is suitable for modeling spatial and non-spatial data in an organized way compatible with the Unified Modeling Language (UML[6]).

Figure 4.3 shows the OMT-G diagram representing the 2013/TIGER Line dataset main classes, including the generated collections for the urban network query set (Subsection 4.3.3). The TIGER shapefile collection duplicates data in order to provide isolation between classes. This enables for the general public to download only the collections they require, as for example, being able to download the PRIMARYROADS subset without having to obtain the entire EDGES collection. This duplication of data also increases performance, as for example, when the most used EDGES subcategories are already in a separate table. Such design characteristic excludes the need for a table filter based on the EDGE attributes in order to verify if that EDGE object is a PRIMARYROAD or a LINEARWATER.

[6]UML: `http://www.uml.org/`

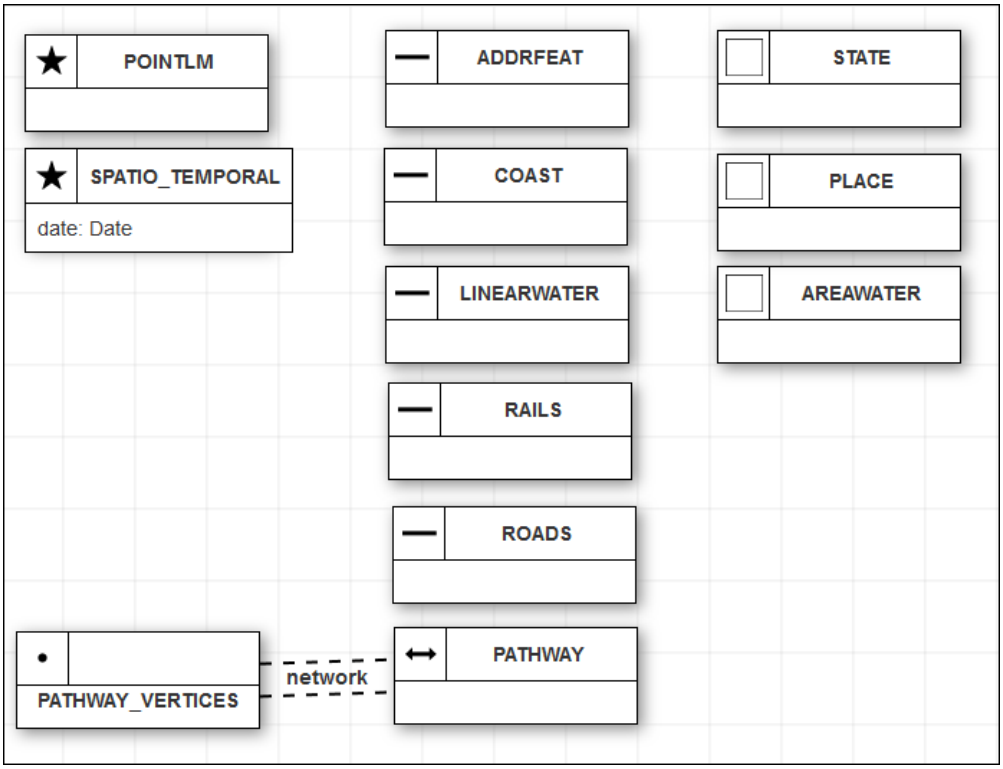


Figure 4.3: TIGER Line OMT-G main classes diagram.

4.3.2 Mapping Spatial Objects

When mapping spatial objects into DBMS, the relational systems usually end up with normalized tables (with some exceptions focusing on either performance or redundancy of data) and spatial columns, which is pretty much straightforward, as demonstrated in PostGIS. The attributes of spatial objects are translated into columns, and the spatial feature itself is a geometry type column. Objects of the same class are stored in their respective tables and the relationship between spatial objects is calculated dynamically through spatial functions and operators. The same does not apply to NoSQL DBMS, for which there might be more than one way to store multiple sets of data with relationships between them.

Considering the example of scientific papers and authors: In a relational DBMS, a usual mapping simply allocates one table regarding **papers**, another one **authors** and a third junction table. This junction table columns' are the primary keys from both aforementioned tables in order to link the papers with their authors. When moving to the document-oriented DBMS (such as MongoDB in our study), there is the possibility of embedding one of the two sets of data inside the other, based on their relationship. Such mapping results in two possible structures: one where the author document contains every paper he has published, and the other where every

paper contains its respective authors. It is important to notice that in both structures, there may be data duplication as the same paper may appear in more than one author document, as well as an author may be inserted in multiple paper objects.

Such problems led us to a common design implementation that focuses on avoiding document embedding, as it makes data alteration rather slow and might increase document size and data duplication. Specifically, the implementation allocates both **authors** and **papers** into two separate collections, with no document embedding. This example applies to the spatial attributes as well, in a way that it requires to dynamically calculate the relationship between two geospatial attributes.

Now focusing on the graph-oriented DBMS, every data item must be translated into either node or relationship attributes. Hence, every spatial object is added as a node to the graph. Nodes are then included in the spatial index through graph relationships. One exception regards the urban routing scenario, where spatial objects belonging to the EDGES collection became relationships as we will explain next.

4.3.3 Graph Mapping for Urban Routing

As detailed in Section 4.6, only PostgreSQL with pgRouting and Neo4j provide graph and network features. With that in mind, we have generated a fully traversable graph representing the United State's automotive network. Specifically, objects that are in the EDGES table were filtered to produce the PATHWAY table. This data collection represents the same spatial attributes as the table ROADS, but the latter is not a fully traversable graph as it does not have road segments, rather full road linestrings. After filtering those objects from the EDGES table, we perform the pgRouting function for creating nodes based on those road segments. Those nodes translate into road intersections represented by the data collection PATHWAY_VERTICES_PGR. With both collections processed and indexed, we are able to perform graph operations, considering target, source and length of each EDGE (road segment).

Now regarding Neo4j, the same datasets were created, but with an important difference. The road segments are not imported as nodes into the graph, but rather as relationships. This procedure is not directly supported by Neo4j and we performed it with a simple script that treated those road segments as relationships between two nodes (road intersections). Then, a graph represents the same network (as in PostgreSQL) built over Neo4j without its spatial extensions. Chapter 5 tests all architectural decisions as we evaluate each DBMS's performance.

4.4 Data Loading

During data loading, the dataset is read, processed and stored in disk. As mentioned earlier, we use the TIGER/Line vector dataset, which provides a large workload similar to many real world spatial databases, with enough data volume to represent a good spatial big data sample. The dataset contains more than 400 million objects, which requires efficient data loading. DBMS restoration and backup are a crucial part of any application's database. Object coordinates in the dataset are originally encoded using the North American Datum of 1983 (NAD83), which we converted to the coordinates in the World Geodetic System 1984 (WGS84), the data used by the Global Positioning System (GPS). The conversion to both WGS84 and GeoJSON (used by MongoDB) was performed using the Geospatial Data Abstraction Library[7].

All three DBMSs provide their own import tool (*shp2pgsql* in PostgreSQL, *mongoimport* in MongoDB and Neo4j's *ShapeFileImporter*). *Shp2pgsql* performed very well right out of the box, translating shapefiles into PostgreSQL data relatively fast and without issues. However, the import tools provided by MongoDB and Neo4j were much slower, thus new tools were implemented to achieve a more reasonable import time. Also, as the import procedure is not executed as frequently as regular querying, data modification and backup routines, it is not part of the final performance evaluation. Detailed information about the improved data loading tools are described in Section 4.5.

After completing the dataset import, spatial indexes are created for all geometry fields. Indexes are also created for all attributes used in queries, such as road type (rural and local roads, highways and interstates) for urban routing, and point types for nearby points of interest. For MongoDB and Neo4j, our import tools generated the exact same database, with the same number of objects, with varying disk usage. For Neo4j, we were unable to compare the default import tool results, as its data importer was too slow: we canceled it after 72 hours of processing.

The disk space used by each DBMS compared to original files (shapefile and geojson) points out interesting behaviors. First by looking at data usage, it is clear that storing data as tables is much less expensive than documents and graphs. This happens because both graph and document systems store the key-value pair for each attribute, which is not necessary on relational systems as the key (column name) is not replicated on every object. Now regarding the disk space consumed by indexes, the generalized search tree (GiST) used by PostgreSQL is not as efficient as Neo4j's graph strategy. MongoDB's index disk consumption reflects the Geohash associated with a

[7]GDAL 1.11.0: `http://www.gdal.org`

Table 4.3: Dataset disk consumption after data loading.

Storage	Data	Indexes	Total
ShapeFile	136 GB	-	136 GB
PostgreSQL	132 GB	65 GB	197 GB
GeoJSON	224 GB	-	224 GB
Neo4j	271 GB	18 GB	289 GB
MongoDB	412 GB	32 GB	444 GB

Table 4.4: Dataset file import time for the original and implemented tools.

Import Tool	Original Import Time	New Import Time
PostgreSQL	26 hours	-
MongoDB	70 hours	33 hours
Neo4j	Interrupted after 72 hours	56 hours

space-filling curve, more space than Neo4j but less than PostgreSQL. The disk space used by the dataset in each platform, as well as compacted shapefiles and structured files, are presented in Table 4.3.

4.5 Improvements on Data Loading

As aforementioned, the default tools spent too much time on data loading. Therefore, we have improved their performance with one solution for MongoDB's *mongoimport* and another for Neo4j's *ShapeFileImporter* class, as explained next.

4.5.1 Improving Mongoimport

Mongoimport has issues regarding processing time and import speed. Its performance is also unstable and degrades fast when large amounts of data are converted from JSON to MongoDB's native binary documents (BSON). The documentation[8] instructs users to run multiple *mongoimport* instances, as this tool is single threaded; but even then it is not very efficient. The issue with running multiple instances of *mongoimport* is that parallel insertion alters the order in which the objects are being stored. This may influence the index structure and possibly cause collection fragmentation over multiple data files, which is unacceptable in a performance evaluation study. Overall, the import tools suffer from slow data serialization and processing time.

Our solution is to lift the task of document serialization from the DBMS. Instead, we propose to perform it in a separate thread, so as to speed up the import time and use more than one processor core. This simple strategy allows the import tool to serialize an object while another is being inserted in the DBMS, thus providing a pipeline that is considerably faster than the serial solution from *mongoimport*. Indeed,

[8]MongoDB Documentation: `http://docs.mongodb.org/manual/`

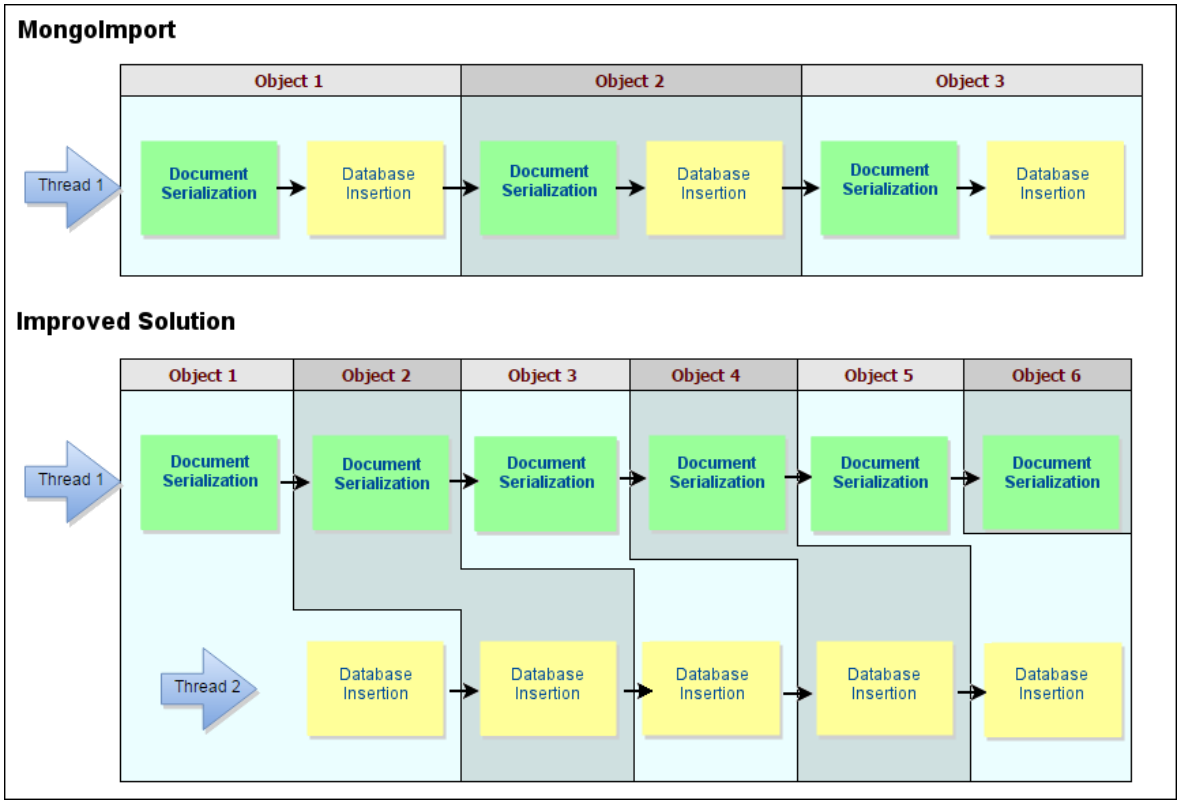


Figure 4.4: Thread schematic comparing Mongoimport with the new proposed solution.

such solution proved to be better by importing GeoJSON documents roughly 50% faster than *mongoimport*, as presented in Table 4.4. If document order is not relevant, import speed could be improved even further by increasing the number of threads serializing documents concurrently. In the event of a machine cluster, increasing the number of machines would also speed up database insertion. Figure 4.4 thread diagram represents how the new solution operates. Nevertheless, this time is still 27% slower than *shp2pgsql*.

4.5.2 Improving Neo4j Load Strategy

When loading data into Neo4j, the spatial extension builds an R-Tree over the graph structure to index spatial data. However, it is not possible to add spatial data to the database without adding it to the spatial index first, thus making the loading process very slow. In other words, Neo4j has to maintain most of the index in memory and keep modifying it as new data is added.

Here, we implement a much faster batch insertion strategy. It consists of indexing data only *after* the entire insertion process is complete. However, this strategy requires that inserting formatted data into raw graph nodes and relationships happens without

Table 4.5: The main features, the spatial data type returned and what the evaluation measures for each query group.

Query Group	Spatial Data Type	Evaluation Focus
Nearby Points of Interest	Vector	Index efficiency and point-to-geometry processing
Map View	Vector	Memory management and locality of reference
Urban Routing	Network	Disk access strategies and raw processing power
Position Tracking	Spatio-Temporal	Persistence strategies and index management

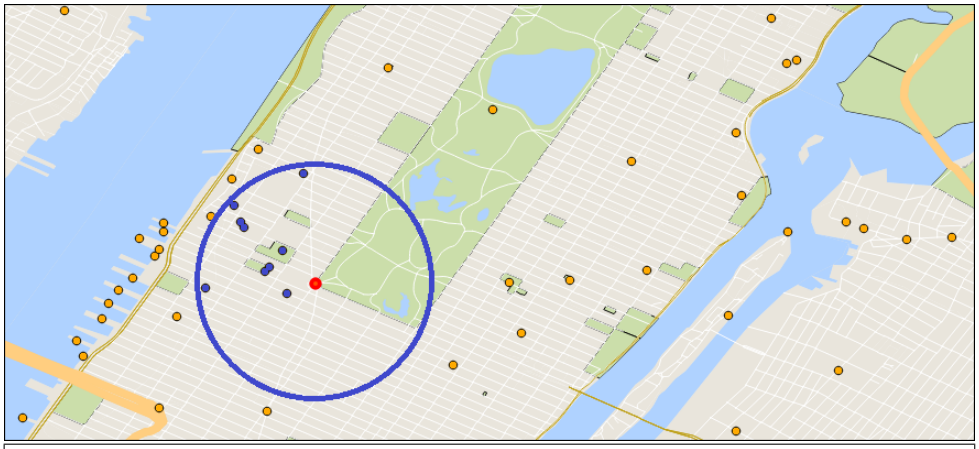
Neo4j Spatial. Hence, our Neo4j import tool adds the nodes and edges to the Neo4j standard graph structure. Later, the nodes and relationships stored in the DBMS are added to the spatial layers (indexes), now returning to use the Neo4j Spatial Extension. This solution is faster than Neo4j's, as presented in Table 4.4 (even though it is still much slower than the others).

4.6 Spatial Queries

Here, the set of queries is designed based on features from mobile GIS and spatial applications. In other words, they are specifically based on functions commonly used in mobile applications, such as Google Maps, Waze, Foursquare, etc. There are four main query groups: nearby points of interest, map view, urban routing and position tracking, as described in Table 4.5 and detailed next.

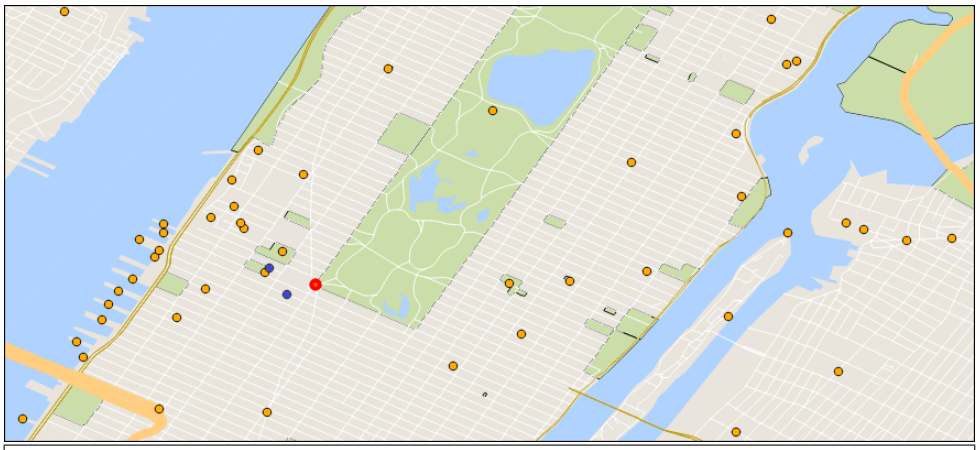
Nearby Points of Interest. This group contains two of the most common types of queries performed by users when looking for nearby points of interest (POI). The first is a query that searches for POIs closer than a given distance from a reference position (usually the device's position). One example of such a query is: "Find gas stations closer than 5 kilometers from my workplace". This group tests mainly the DBMS's index efficiency and its point-to-geometry processing power. Figure 4.5 illustrates a POI within radius visual representation with the respective PostgreSQL query.

The second is to query for the **k-nearest neighbors** (k-NN): a search for one or a group of the closest objects from a given location. Examples are: "Where is the closest bus stop to my dentist appointment?" and "What are the two closest point landmarks near Columbus Circle in Central Park?". Figure 4.6 exemplifies a k-NN visual representation with the respective PostgreSQL query.



```
SELECT * FROM POINTLM plm
WHERE ST_DWithin(plm.geom,ST_GeomFromText
('POINT (-86.7018143434 32.4798804232)',4326),1.0 );
```

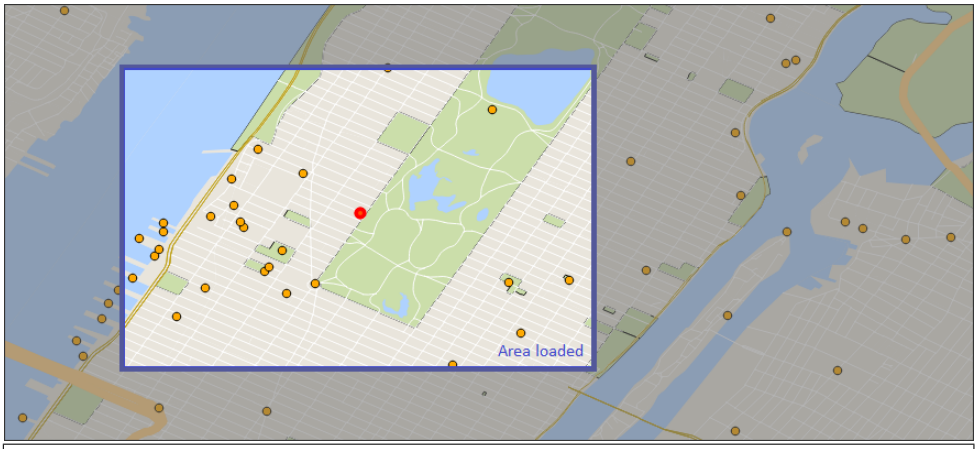
Figure 4.5: Example of nearby POI within radius and equivalent PostgreSQL query.



```
SELECT * FROM POINTLM plm
ORDER BY plm.geom <-> ST_GeomFromText
('POINT(-86.7018143434 32.4798804232)', 4326) ASC LIMIT 1;
```

Figure 4.6: Example of nearby POI KNN and equivalent PostgreSQL query. The red mark is the device's position and the blue marked locations are the possible answers.

Map View. This group simulates interactive user browsing, zooming and panning. The zoom usually ranges from a city block (150 meters wide) to hundreds or thousands of kilometers. This group tests how well the DBMS memory manager handles the locality of reference on its indexes. An optimized indexing and caching implementation may perform better when loading spatial objects that are close to the previously loaded ones. It is also important to evaluate how the DBMS handles queries in multiple collections/tables/layers at the same time, requiring access to multiple indexes and



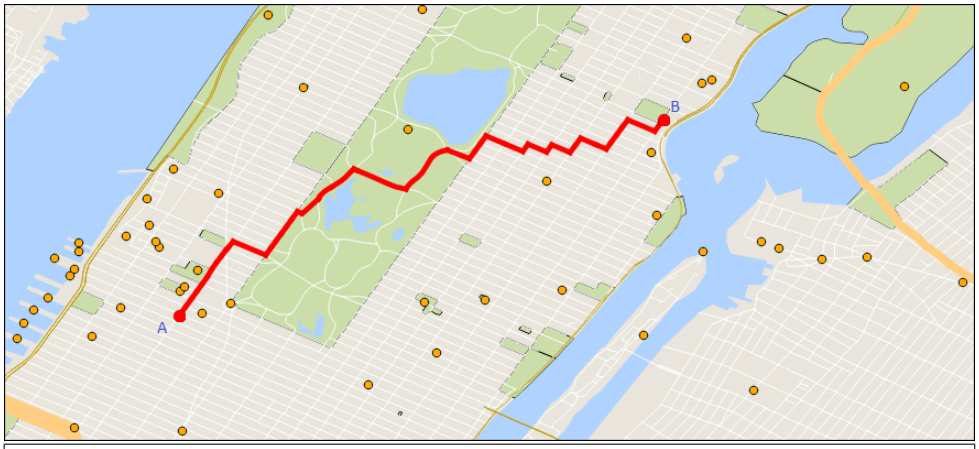

```
SELECT * FROM FACES fcs
WHERE ST_Intersects(ST_GeomFromText('POLYGON
((-87.2018143434 31.9798804232, -87.2018143434 32.9798804232,
-86.2018143434 32.9798804232, -86.2018143434 31.9798804232,
-87.2018143434 31.9798804232))', 4326), fcs.geom);
```

Figure 4.7: Example of map view and equivalent PostgreSQL query. The user's position is represented in red and the loaded results are highlighted within a rectangle for the "Area loaded".

database files. Figure 4.7 illustrates a visual representation of the returned objects with the respective PostgreSQL query.

Urban Routing. This group plays a very important role nowadays, as applications like Google Maps, Bing Maps and Waze already provide a routing function. Here, the test queries compare the systems' response to graph-oriented operations and determine how well their shortest path algorithms behave. The urban routing algorithms are not performed over long distances as there are many solutions to filter graph edges that would not likely be traversed by the algorithm, or to avoid extensive calculations.

For example, when Google Maps queries for the shortest path between two very distant points (e.g., Seattle and New York), it uses heuristics to focus on highways and roads, then ignoring less important paths that are not close to both the source and the destination. Such priority setup reduces the number of edges considered by the shortest-path algorithms, thus making the processing time faster without altering much the resulting path. We also use the same principle: the further the source and destination points are, the more likely the algorithm is to avoid using local neighborhood roads, city streets, secondary roads and finally primary roads. The minimum and maximum distances provided to the routing queries are based on data from the




```
SELECT seq,id1 AS node,id2 AS edge,cost,geom
FROM pgr_dijkstra
('SELECT r.gid AS id, r.source, r.target, st_length(r.geom)
AS cost, r.geom FROM PATHWAY r, PATHWAY_VERTICES_PGR v1',
36123091,36112650, FALSE, FALSE)
AS di JOIN PATHWAY pt ON di.id2 = pt.gid
```

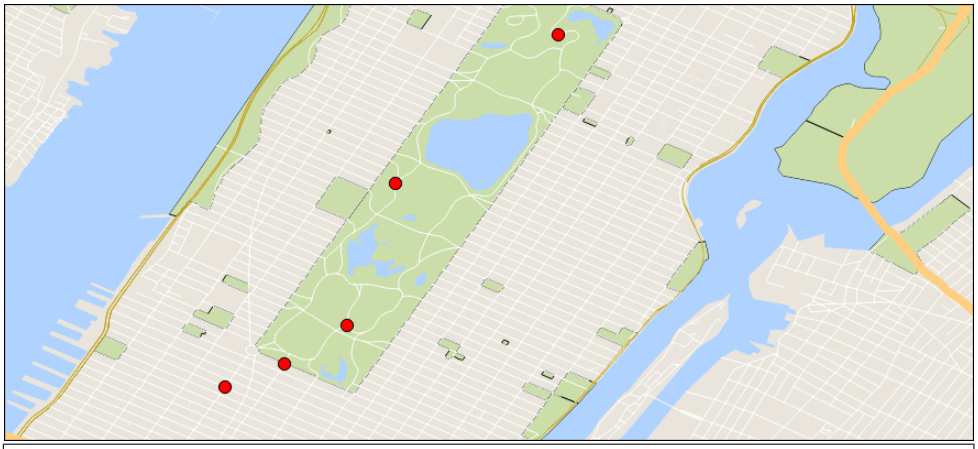
Figure 4.8: Example of calculated shortest path between points A and B and equivalent PostgreSQL query.

National Household Travel Survey[9] (NHTS) – an effort sponsored by the Bureau of Transportation Statistics (BTS) and the Federal Highway Administration (FHWA) to collect data on both long-distance and local travel by the American population. Figure 4.8 illustrates a calculated shortest path visual representation with the respective PostgreSQL query.

Both PostgreSQL's pgRouting and Neo4j provide Dijkstra's [16] and A* [25] algorithms to perform the search for the shortest path within a network. In this study, we consider the Dijkstra's algorithm in both DBMS, since it does not rely on heuristics like the A* (i.e., it is less susceptible to variations and particular best/worst case scenarios).

Position Tracking. This is a feature (available in many mobile devices with or without the user's knowledge) that constantly records the user's position in order to provide location-based services (such as traffic management, fleet location, logistics and delivery optimization, remote sensors, historical location heatmaps, etc.) or simply to deliver location-specific advertisement. Therefore, this feature requires frequent storage of point locations. We propose a very simple but efficient way of measuring how well

[9]NHTS :`http://www.rita.dot.gov/bts/sites/rita.dot.gov.bts/files/subject_areas/national_household_travel_survey/index.html`



```
INSERT INTO SPATIO_TEMPORAL(calendar,geom)
VALUES (TIMESTAMP '2014-10-26 23:15:40.63',
ST_GeomFromText('POINT (-83.250147 31.818538)', 4326));
```

Figure 4.9: Recorded position history on the map and equivalent position insertion in PostgreSQL.

a system can trace its clients' locations over time and add them to an existing data collection. As both time and spatial attributes are indexed, index restructuring and file management play important roles. This group of queries evaluate how fast the DBMS can insert new data into an existing collection with previously built indexes. A visual representation of inserted device locations with the respective PostgreSQL command is presented in Figure 4.9.

Chapter 5

Experimental Evaluation

In this chapter, we describe the experimental setup, database configuration, index creation, software and hardware characteristics, parameters, metrics and the performance evaluation. Also, we detail any exceptional characteristics that excluded one of the DBMSs from a spatial query category, including bugs, non-equivalent query algorithms and lack of features.

5.1 Experimental Setup

The experimental evaluation was performed in an Intel Core i7 3770K with 32 GB of non-ECC 1600MHz RAM, 120GB SSD for the operating system and DBMS binary files and a 2TB 7200 RPM hard drive dedicated to the DBMS storage folders. The operating system is Windows Server 2012 64 bits. Only essential tasks and services run during the execution to avoid unwanted system hardware calls and resource concurrency. Regarding DBMS tuning, Neo4j Java heap size was increased for matching the other DBMS natural behavior of occupying all available RAM. Also, spatial indexes were created in all spatial columns (PostgreSQL), fields (MongoDB) or attributes (Neo4j).

We have implemented all evaluation functions in a tool that avoids interfering with the DBMS's performance. Also, all query caching, file buffering and other strategies were handled exclusively by each DBMS. As for specific setups, the Neo4j embedded server runs on Java Virtual Machine (JVM) and was placed in a separate thread. This way, it uses memory mapped buffers and avoids any memory sharing with the evaluation tool, then putting the operating system in charge of its buffers. The drivers utilized are: PostgreSQL 9.3-1102 JDBC driver, MongoDB 2.12.3 Java driver and Neo4j embedded server 2.1.4.

Table 5.1: Number of query executions per query set.

Query Set	Number of single executions
Nearby POI Radius	1,000
Nearby POI k-NN	1,000
Map View	1,000
Map Zoom	1,000
Map Pan	1,000
Urban Routing	20
Position Tracking	500,000

5.2 Parameters and Pre-evaluation

In order to setup a meaningful set of queries, we have defined specific ranges that limit the values generated for each group of queries. We also expect and reinforce variety and quantity of the resulting sets, in order to better evaluate how each DBMS handles its spatial indexes in both memory and disk. Each query group has a number of executions per set, as demonstrated in Table 5.1. Also each query is started by its own thread at a random time interval (which enables multiple queries running at the same time). This distribution of query executions per set mimics the proportion of procedures performed by mobile devices over these mobile systems' servers. These queries consider the parameter value ranges presented in Table 5.2 and explained as follows.

- For the **Nearby POI Radius-based** group, the maximum radius is 1.0 degree (approximately 110 kilometers) and minimum is 0.01 (approximately 1 km). Then, each **k-NN** query varies between 1 and 10 nearby points. The minimum (1 km) value is less than the Americans' average daily walking distance [4] (4.11 km), and the maximum (110 km) is double the distance used in the Urban Routing query group.

- For the **Map View** group, the query window is a 16:9 rectangle (to simulate the mobile user's screen format) with width and height using a minimum of 0.0013 degree (approximately 150 meters, a city block) and maximum of 1.0 degree (110 km, double the distance used in Urban Routing).

- The **Urban Routing** queries have limited maximum distances between origin and destination, where the minimum distance is 10 meters (the shortest road edge available) and the maximum is 0.5 degree. According to the NHTS, this value is the 90th percentile of the American Commute Distance (One Way), and corresponds to 56 kilometers or 35 miles. The distances may differ slightly as degrees conversion into kilometers vary according to the latitude.

Table 5.2: Evaluation parameters.

Query Group	Minimum	Maximum
Nearby POI Radius	0.01 degree	1.0 degree
Nearby POI k-NN	1 place	10 places
Map View	0.0013 degree	1.0 degree
Urban Routing	0.0001 degree	0.5 degree
Position Tracking	100 points	100,000 points

- During the **Position Tracing** evaluation, the spatio-temporal object is composed by the object id, its date of creation and a randomly generated point geometry within the urban routing network. The number of insertions performed simulates a daily journey (random routes) of an object traveling between 1 and 60 km/h, which implies a number of points ranging from 100 to 100,000 – depending on the distance between consecutive points. The point range is associated with the minimum android GPS poll interval (every 15 minutes, or 96 times a day), to a maximum 100,000 requests a day, when the device is actively using a spatial application (i.e.: Google Maps or Waze).

Finally, for each group of queries, we performed a pre-evaluation of each DBMS in order to verify any inconsistency towards a fair comparison. Specifically, for the **Urban Routing** group, MongoDB is not considered because the evaluated version (2.12.3) does not provide any network-related data handling. Also, given that Neo4j is a graph oriented database, its shortest path algorithms are part of Neo4j's core; i.e., it does not require a spatial extension to perform the calculation, using road segments as edges and intersections as nodes.

Then, for the **k-NN** queries, Neo4j Spatial results differed greatly by returning empty collections or incomplete result sets. Since Neo4j Spatial is open source, a code review pointed out that its query method does *not* meet the standard k-NN implementations: in each query, it uses a density estimation to calculate the size of the polygon; then a within search is performed to match elements inside the generated polygon. Therefore, Neo4j's k-NN does not perform the claimed query. Its results are not compatible with the other two DBMSs, which led to a code review in order to understand its query behavior. The *org.neo4j.gis.spatial.SpatialTopologyUtils* module contains the method *createEnvelopeForGeometryDensityEstimate*, whose density estimate method is shown in Algorithm 1. After the envelope is calculated, a within polygon search is performed as in Algorithm 2. Nonetheless, this approach does *not* work as a nearest neighbor search, because there is no guarantee that the density estimation will cover the number of objects requested. Finally, as it is performing a within polygon and not a k-NN search, we do not consider Neo4j in the k-NN evaluation group.

Algorithm 1 Create envelope for geometry density estimate.

```
procedure createEnvelopeForGDE(Layer layer, Coordinate point, int limit)
    if limit < 1 then
        return new Envelope(point)
    end if
    int count = layer.getIndex().count()
    if count > limit then
        return createEnvelopeForGDE(layer, point,(double) limit / (double) count)
    else
        return Utilities.fromNeo4jToJts
        (layer.getIndex().getBoundingBox())
    end if
end procedure
procedure createEnvelopeForGDE(Layer layer, Coordinate point, double fraction)
    if fraction < 0.0 then
        return new Envelope(point)
    end if
    Envelope bbox = Utilities.fromNeo4jToJts
    (layer.getIndex().getBoundingBox())
    double width = bbox.getWidth() * fraction
    double height = bbox.getWidth() * fraction
    Envelope extent = new Envelope(point)
    extent.expandToInclude(point.x - width/2.0, point.y-height/2.0)
    extent.expandToInclude(point.x+width/2.0, point.y+height/2.0)
    return extent
end procedure
```

Algorithm 2 Using the generated envelope area to perform a within polygon search.

```
procedure startNearestNeighborSearch(Layer layer, Coordinate point, Envelope searchWindow)
    return start(layer, new SearchIntersectWindow
    (layer, searchWindow)).calculateDistance
    (layer.getGeometryFactory().createPoint(point))
end procedure
```

5.3 Evaluation Metrics

Regarding evaluation metrics, we consider the query execution time and number of vertices returned by the query. Specifically, the number of vertices is the sum of the vertices of the polygons of all objects, documents or tuples returned by the queries. We also propose the *vertices per second* (v/s) measurement as a performance indicator, observing that spatial data is best analyzed when considering the number of points that form a geometry. Usually query processing over a more complex geometry or spatial object takes considerably more processing power than operations over singular points. The metrics derived from the main attributes are the average query time and vertices per second throughput, which is defined as follows.

Vertices Per Second: Let t be the time in seconds spent to evaluate a query, n the number of spatial objects retrieved, and v the number of vertices of each object. The *vertices per second* is the sum of all object vertices returned by the query divided by its execution time:

$$VerticesPerSecond = \frac{\sum_i^n v_i}{t} \tag{5.1}$$

5.4 Performance Evaluation

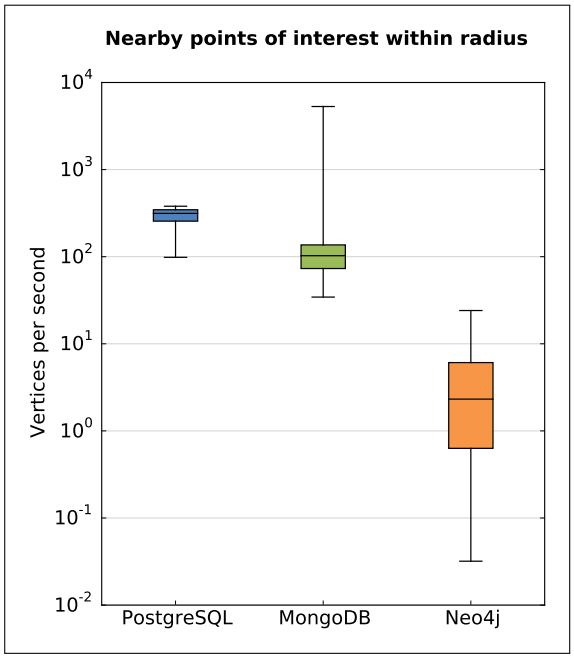
In this section, we evaluate the DBMS for spatial data. As aforementioned, the evaluation metrics are the number of vertices per second (v/s, the higher the better) and execution time (the lower the better). The limits of the box represent the first quartile (designated Q1) or the 25th percentile, and the third quartile (Q3) or the 75th percentile. The ends of the whiskers represent the 5th and 95th percentiles. Detailed percentile information about the next experiments can be found in Appendix A.

5.4.1 Nearby Points of Interest Within Radius and k-NN

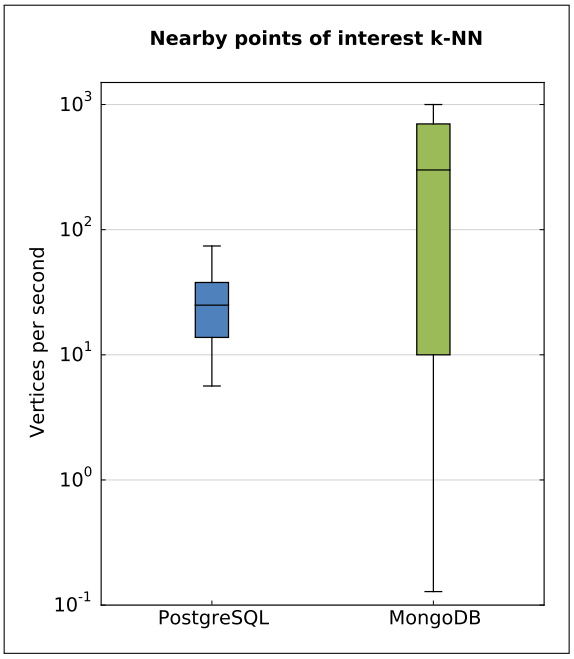
We start the performance evaluation with the nearby points of interest (POI) within a given radius and k-NN queries. Figure 5.1a shows the results for Nearby POI within Radius with the distribution of number of vertices per second for each DBMS. Note that PostgreSQL has the best median throughput and its standard deviation is small. Then, MongoDB has a relative slower median than PostgreSQL, with maximum scores higher than PostgreSQL and minimum nearly equal. Neo4j falls into another performance level with a maximum vertices per second score very close to the other two DBMS's minimum, which is recurrent in other experiment categories as explained later on.

Figure 5.1b shows the vertices per second for the **Nearby Points of Interest k Nearest Neighbors** query set (there are no results for Neo4j due to its k-NN implementation and query results, as explained in Section 4.6). In this group, MongoDB's performance surpasses PostgreSQL by three times more vertices output. This is because both approaches are different, as explained next.

The k-NN approaches used by all three DBMS are different. PostgreSQL searches directly over an R-tree and uses the '$<->$' operator, which returns the distance between two points (it uses the geometry centroid for polygons to speed up the calculation). Then, MongoDB searches on multiple iterations over concentrically growing circular polygons until the last neighbor is found. Overall, even though the PostgreSQL approach is less expensive memory-wise, MongoDB's index for points associated with growing concentric circles proves to be faster as there are no distance calculations to be made between each point (except on the last concentric circle iteration). Neo4j's implementation is a poor approximation with bad performance and usually no objects are retrieved in low density areas, thus being excluded from this experiment group.



(a) Results for nearby POI within radius.



(b) Results for nearby POI k-NN.

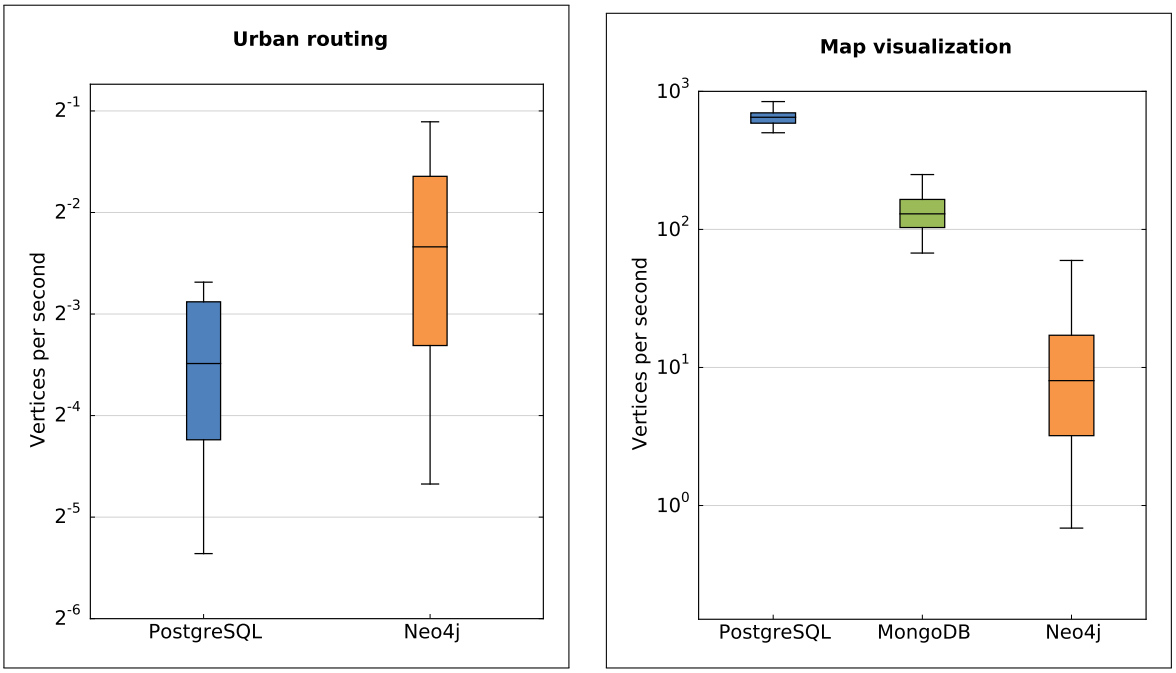
Figure 5.1: Distribution of query execution results (v/s).

5.4.2 Urban Routing

Figure 5.2a shows the results for Urban Routing queries, where the number of vertices per second processed by Neo4j is roughly twice those of PostGIS' pgRouting. Such results reveal Neo4j's greatest comparative strength: calculating the shortest path between two points in a network, as expected for a graph-oriented database management system. Also, such results may indicate that the performance bottleneck for other types of queries might be caused by Neo4j's Spatial Extension, rather than the graph engine itself, as the shortest path query is a native feature of Neo4j and does not require an R-tree-over-graph index structure and any other spatial extensions. Another important fact is that urban routing queries are much slower than the other groups, thus supporting our evaluation focus on disk access and CPU processing power required by this query group.

5.4.3 Map View

The difference in performance is even clearer in this set of experiments. Figure 5.2b shows the results: each DBMS is separated from the other by a factor of *ten*. As a relational DBMS, PostgreSQL outperforms the other DBMSs at combining the multi-table query results while using the spatial index. MongoDB struggles to deal with

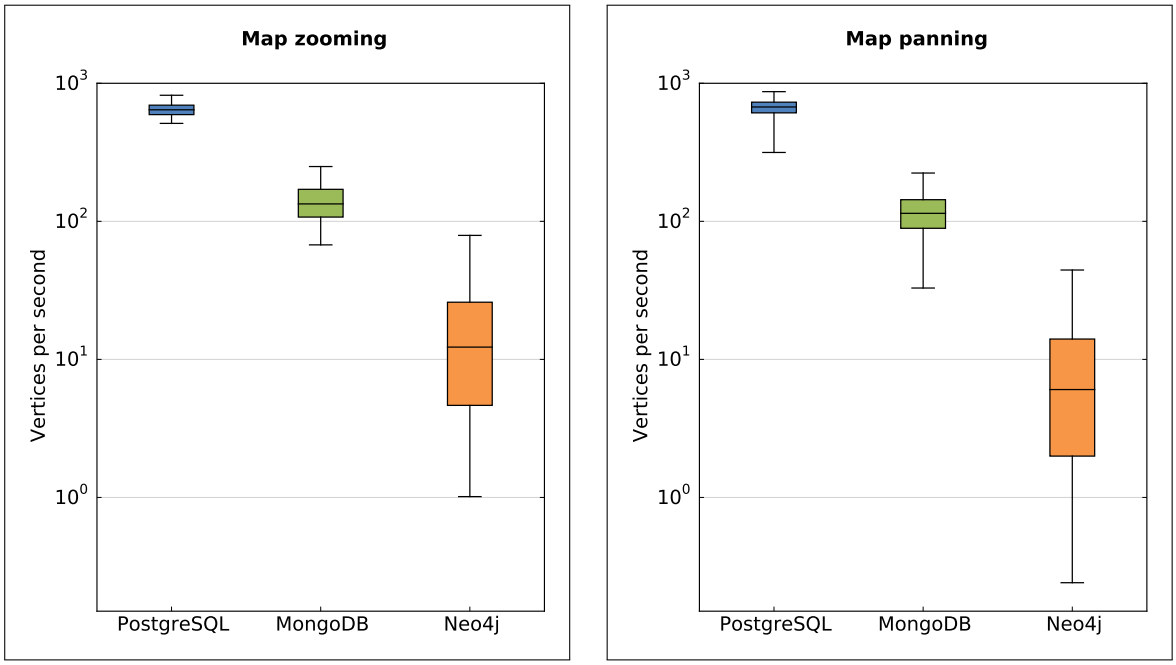


(a) Results for urban routing (log2). (b) Results for map visualization (log10).

Figure 5.2: Distribution of query execution results (v/s).

multiple collections at the same time, as its memory management strategies focus on keeping the entire indexes on memory as much as possible. Such approach might not be ideal when querying a collection forces another collection's index out of the RAM. Neo4j again performs poorly, mostly because its R-tree index implementation over the graph structure forces the index to be completely loaded into memory as a graph (spatial layer), which is slower.

Zooming and panning are also part of the Map View group. In a map view, our analysis evaluates how the accessed data and indexes are used in the next queries. It is expected that any indexes previously loaded into memory by the Map View query will remain available for the zooming and panning. Also, if any DBMS provides object caching, those objects will be expected to be retrieved faster. Hence, Figure 5.3a and Figure 5.3b show a throughput increase, when compared with the Map View, by zooming and panning a previously accessed portion of data. The results show the performance of PostgreSQL as being constant at a rate close to 800 vertices per second, while MongoDB reached little over 100, and Neo4j barely achieves 10 vertices per second. It is possible to notice that the minimum score for all three DBMSs falls greatly during panning. Such result indicates that the query selected a portion of the dataset that was not accessed by either viewing or zooming the map, resulting in (expensive) disk operations rather than memory accesses.



(a) Results for map zooming.

(b) Results for map panning.

Figure 5.3: Distribution of query execution results (v/s).

5.4.4 Position Tracking

Now, we analyze how the systems manage insertions into both spatial and data indexes. Figure 5.4 shows the average vertices per second inserted into the database. Whereas PostgreSQL still maintains a relatively constant insertion time, both MongoDB and Neo4j oscillate between fast insertions and slow ones, that are almost 100 times slower than PostgreSQL's. As all three DBMS have different insertion strategies, the insertion v/s results have high variability. Detailed percentile information about the distribution can be seen in Appendix A. These results indicate that the PostgreSQL solution is the fastest on average, and both MongoDB and Neo4j implement a delayed insertion mechanism. Specifically, MongoDB considers replica sets and log registers for writing operations. The first one guarantees the write operation will propagate insertions to all replica sets (when there is any). The second one ensures durability by confirming that any insertion operation has been logged into the on-disk log. As we are assessing single machine performance, and MongoDB's strategy is focused on scaling with machine clusters, future experiments may end up with different results. On the other hand, Neo4j uses a commit interval parameter: if the number of insertions is equal or greater than the commit interval, all modifications are pushed to the disk and the commit interval is set to zero again. Such strategy's behavior can be seen in Figure 5.5, as explained next.

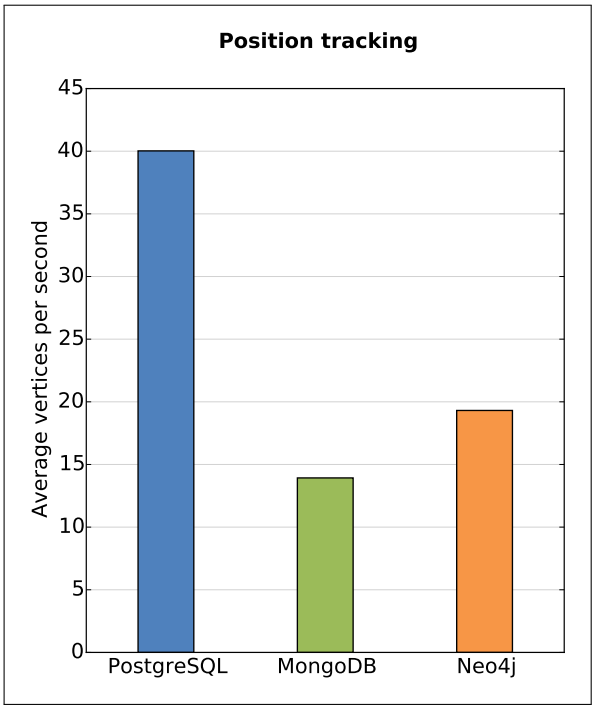


Figure 5.4: Results for average vertices inserted per second.

5.5 Relative Performance Summary

The relative performance is calculated by dividing the DBMS query execution time by PostgreSQL's, the oldest (and arguably more mature) of the three. PostgreSQL is taken as a reference with normalized time of 1. Other DBMS values indicate how much faster or slower they are in comparison. Figure 5.5 shows the normalized execution time for samples of 20 queries per group.

According to our evaluation results, the DBMS with best overall performance is PostgreSQL. Even though our dataset considered a sample when compared to real big data collections, there is evidence to support that this mimics one of the clusters' machines in terms of processor, memory and disk usage. As a more established product, PostgreSQL with PostGIS is still the reference when handling spatial data, even if its relational-oriented characteristics are not built for big data. Nonetheless, despite PostgreSQL performance being consistent and superior *overall*, its horizontal scalability is supposedly not as easy to achieve as the two NoSQL DBMS tested. Specifically, PostgreSQL offers many scaling and table partition tools. However, such tools require implementing and configuring many variables, as well as managing and tuning the existing partitions and analyzing the gains obtained. This whole task is very time consuming, and may not compensate the extra effort.

MongoDB manages to stay relatively close to PostgreSQL in most groups, and

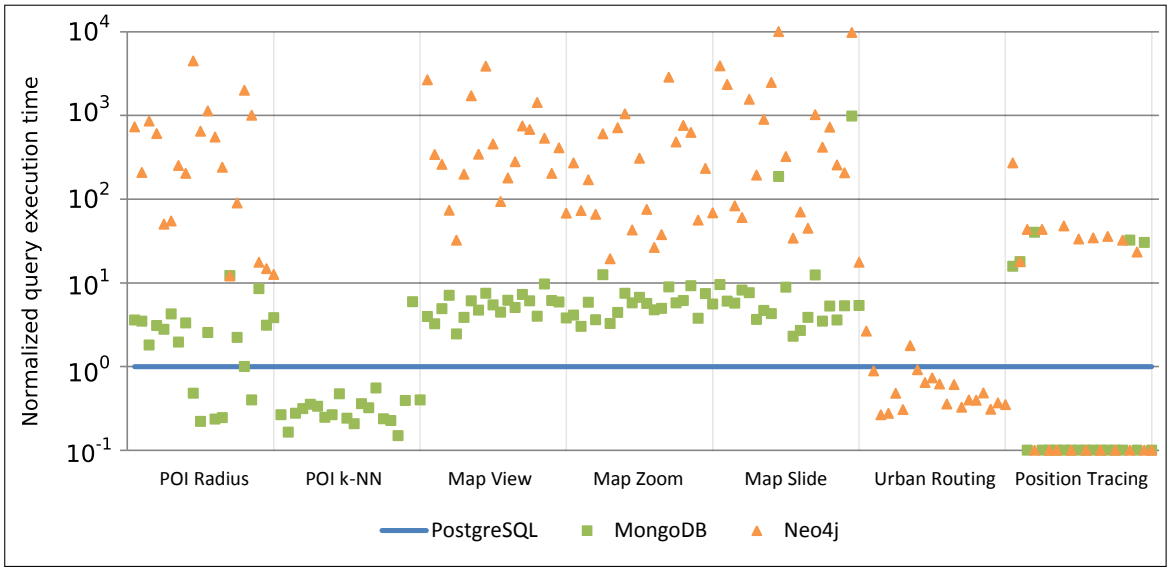


Figure 5.5: Relative performance summary.

outperforms it during the POI radius and k-NN. Such results show that its growing concentric circles implementation can be as fast as PostgreSQL's. Neo4j's overall performance was weaker, but being the youngest system considered and graph-oriented, it is somehow expected not to perform well under circumstances that favor tree-indexing and very efficient disk and memory management. Moreover, being a graph DBMS, Neo4J has shown good performance for Urban Routing queries by outperforming PostGIS's pgRouting when using its core shortest path implementation.

We now take a further look at the weaker Neo4j performance by utilizing Resource Monitor[1] and the Java VisualVM[2] to investigate what may have caused such degraded performance. Figure 5.6 depicts the time spent at several methods during Neo4j's query executions. While the disk used in this experiment is capable of an average reading speed of 152 MB/s, the total disk I/O during Neo4j's benchmark peaked at 6MB/s. Such results indicate a large amount of seek and random accesses, even though its disk usage was consistently near the 100% mark. The figure also points out the methods that consume most of the CPU time, which belong to the file storage management system and to the cache system. Thus, there is evidence to support that the DBMS, while using its spatial extension, has an underperforming file management interface.

In summary, our results show that PostgreSQL with PostGIS is a DBMS that works well under multiple scenarios and provides the most spatial features. MongoDB is almost as fast as PostgreSQL in some cases, performs well in the k-NN scenario and

[1]Windows Server Resource Monitor: `http://msdn.microsoft.com/en-us/library/ms191246.aspx`

[2]Java VisualVM: `http://visualvm.java.net/`

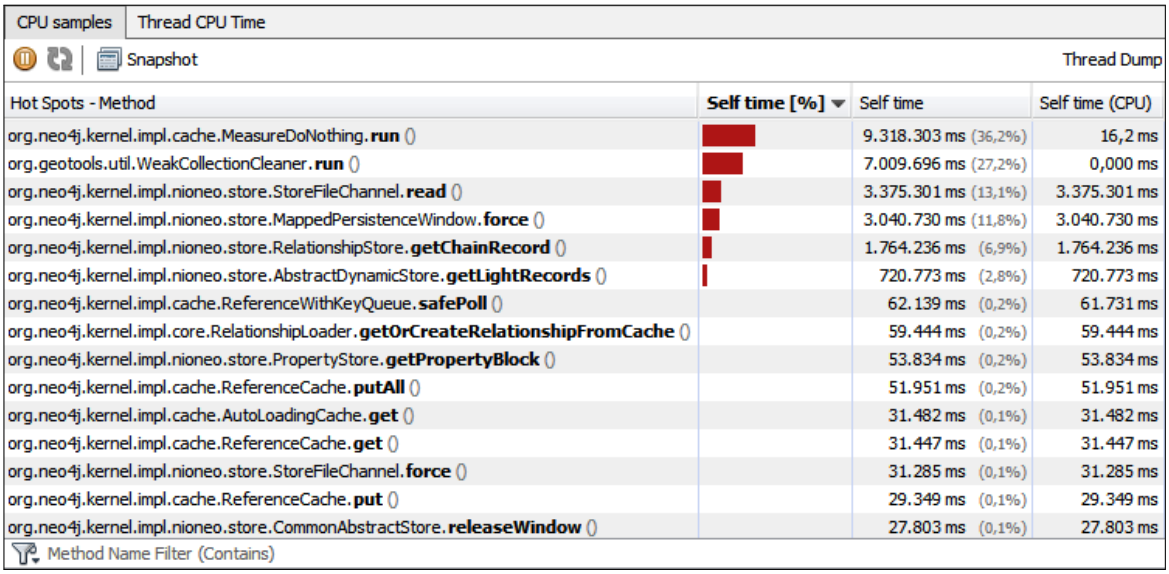


Figure 5.6: Neo4j JavaVM CPU Sampling.

is easy to scale horizontally, but it lacks many spatial features, including networking. Neo4j has a slow performance when compared with the other DBMS but it has a very good shortest path network implementation and should be the best option when performing routing queries.

Chapter 6

Conclusion

Evaluating and benchmarking spatial DBMS performances is not as simple as evaluating relational DBMS, because spatial attributes are much more complex to handle than strings, numbers and other relational data types. While several system's architectures provide different levels of performance and features, it is imperative to limit the data profile in which an analysis is made. Even in a very specific scenario (i.e., mobile applications), our evaluation achieved very heterogeneous results at both DBMS and query group levels.

Current evaluation tools and methodologies such as the TPC-C (even though it does not evaluate spatial attributes) and Jackpine focus on relational models, systems and queries. Both TPC-C and Jackpine evaluate the performance based on the time spent on each query and such data is converged into a final score composed by each query's specific weight. Also the queries proposed by Jackpine reflect a general use for a GIS, rather than focused on a service. This differs greatly from the GIS that handle spatial big data, such as Google Maps, Waze and others.

Our experimental evaluation considered 1,000 executions for each vector based query, 20 for network and 500,000 for spatio-temporal recording. With such queries, we simulate a typical mobile application usage throughout the day, with the routing queries being executed less, map visualization being more frequent, and position tracking with much more executions. This combination mimics current mobile devices that continue to produce and upload spatial information into these systems even without the user's interaction. By running this query combination and allocating each one to its own execution thread, our evaluation is able to reproduce combinations between high/low volume and high/low frequency of insertions.

This study compares the performance of different data models, each represented by one product. Such comparison gives evidence of which model to use based on the

desired scenario, and from that point, choosing the best performing DBMS.

6.1 Summary of Contributions

The first contribution is the given set of parameters and metrics for selecting a spatial DBMS based on real usage scenarios and system features. With such parameters, it is possible to mimic the usage pattern on those systems, as for example, the parameters are based on real statistics about daily commuting, such as the average walking distance and maximum commute distance in a day's interval.

Also, we demonstrate that depending on the type of service provided, a particular data model is more adequate to a specific query, thus maximizing performance but also implying the existence of multiple data models for the same dataset.

Our evaluation methodology was composed by five stages (from data loading to shutdown) and performed 100 queries (20 per each of five groups) over a real geographic dataset. It also considered two new data loading implementations. The overall results demonstrated how a relational DBMS performs under large amounts of spatial data, a resulting scenario that may change under heavy clustering and data distribution. Comparing to existing studies, ours summarized the need for more **specific spatial data evaluation methodologies** and more **data and service-oriented benchmarks**, as well as the proposition of a metric that translates the DBMS capabilities into an **throughput comparison**, rather than just only raw response time.

The results of this dissertation were submitted as a paper to the PVLDB journal.

6.2 Future Work

Ideas for improving and extending this work include:

- **Extend this study into a customizable benchmarking tool.** Define a DBMS score with variable parameters based on the users preferences and needs. Making a benchmarking tool which provides the user the right DBMS to be used based on their preferences instead of a standardized and generalized score.

- **Inclusion of a set of queries and parameters to evaluate machine clusters.** Machine clusters and cloud benchmarking require much more control over the test environment in order to provide reliable scores. By extending the tool's support to be able to set up multiple clusters over machines connected in a network could guarantee load balance. The tool would also be required to be able

to tune in a reasonable way each cluster in order to provide a real and fair comparison among many DBMS.

- **Study other spatial performance metrics not evaluated.** Many attributes could be evaluated separately based on each group of queries. For example when querying for intersecting areas and other polygon resulting queries, the area recovered per second could be also a metric to be used, because if the result has many intersections and new sections of polygons created, much more processing power and memory would be required compared to the nearest neighbor search based on points.

- **Development of a Spatial DBMS which encapsulates multiple data models/mapping strategies.** We have observed that systems that focus on a single model, for example the network model and the urban routing category, usually excel in that particular scenario, but lack performance on the rest. One of the DBMS features could be the possibility of representing and storing the same information associated with many mapping techniques, where we could access for example, the collection of road edges and intersections independently (as JSON documents in MongoDB or tables in PostgreSQL) or as a network (layers in Neo4j).

Bibliography

[1] Akal, F., Böhm, K., and Schek, H.-J. (2002). Olap query evaluation in a database cluster: a performance study on intra-query parallelism. In *Advances in Databases and Information Systems*, pages 218--231. Springer.

[2] Banker, K. (2011). *MongoDB in action*. Manning Publications Co.

[3] Baptista, C. S., de Lima Junior, O. F., de Oliveira, M. G., de Andrade, F. G., da Silva, T. E., and Pires, C. E. S. (2011). Using OGC Services to Interoperate Spatial Data Stored in SQL and NoSQL Databases. In *Proceedings of GeoInfo*, pages 61–72, Campos do Jordão, Brazil.

[4] Bassett Jr, D. R., Wyatt, H. R., Thompson, H., Peters, J. C., and Hill, J. O. (2010). Pedometer-measured physical activity and health behaviors in united states adults. *Medicine and science in sports and exercise*, 42(10):1819.

[5] Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B. (1990). The r*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of ACM SIGMOD*, pages 322--331. ACM.

[6] Borges, K. A. V., Davis Jr, C. A., and Laender, A. H. F. (2001). OMT-G: An Object-Oriented Data Model for Geographic Applications. *GeoInformatica*, 5(3):221–260.

[7] Braganholo, V. and Mattoso, M. (2014). A survey on XML fragmentation. *SIGMOD Record*, 43(3):24--35.

[8] Cattell, R. (2011). Scalable SQL and NoSQL Data Stores. *ACM SIGMOD Record*, 39(4):12--27.

[9] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. (2006). Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. on Computer Systems*, 26(2):article no. 4.

[10] Chen, Y., Raab, F., and Katz, R. (2014). From TPC-C to Big Data Benchmarks: A Functional Workload Model. In *Specifying Big Data Benchmarks*, pages 28–43. Springer.

[11] Clementini, E., Di Felice, P., and van Oosterom, P. (1993). A small set of formal topological relationships suitable for end-user interaction. In *Advances in Spatial Databases*, pages 277--295. Springer.

[12] Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377--387.

[13] Cugola, G. and Jacobsen, H.-A. (2002). Using Publish/Subscribe Middleware for Mobile Systems. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):25--33.

[14] Davis Jr, C., Câmara, G., and Monteiro, A. M. (2001). Introdução à ciência da geoinformação. *INPE, São José dos Campos: São Paulo.*, 22(03):2010.

[15] Davis Jr, C. A. and Fonseca, F. (2006). Considerations from the Development of a Local Spatial Data Infrastructure. *Information Technology for Development*, 12(4):273--290.

[16] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269--271.

[17] Egenhofer, M. J. and Franzosa, R. D. (1991). Point-set topological spatial relations. *International Journal of Geographical Information Systems*, 5(2):161--174.

[18] Floratou, A., Teletia, N., DeWitt, D. J., Patel, J. M., and Zhang, D. (2012). Can the Elephants Handle the NoSQL Onslaught? *PVLDB*, 5(12):1712--1723.

[19] Ghazal, A., Rabl, T., Hu, M., Raab, F., Poess, M., Crolotte, A., and Jacobsen, H.-A. (2013). BigBench: Towards an Industry Standard Benchmark for Big Data Analytics. In *Proceedings of SIGMOD*, pages 1197--1208, New York, NY, USA.

[20] Gou, G. and Chirkova, R. (2007). Efficiently querying large XML data repositories: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(10):1381--1403.

[21] Graefe, G., Volos, H., Kimura, H., Kuno, H. A., Tucek, J., Lillibridge, M., and Veitch, A. C. (2014). In-Memory Performance for Big Data. *PVLDB*, 8(1):37--48.

[22] Greengard, S. (2014). Weathering a new era of big data. *Commun. ACM*, 57(9):12--14.

[23] Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. In *Procs of SIGMOD*, pages 47--57.

[24] Gyssens, M., Paredaens, J., Van den Bussche, J., and Van Gucht, D. (1994). A graph-oriented object database model. *Knowledge and Data Engineering, IEEE Transactions on*, 6(4):572--586.

[25] Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100--107.

[26] Hecht, R. and Jablonski, S. (2011). NoSQL Evaluation. In *Proceedings of IEEE CLOUD*, pages 336–341, Hong Kong, China.

[27] Hellerstein, J. M., Naughton, J. F., and Pfeffer, A. (1995). Generalized search trees for database systems. In *Proceedings of VLDB.*, pages 562--573.

[28] Hora, A. C., Davis Jr, C. A., and Moro, M. M. (2011). Mapping Network Relationships from Spatial Database Schemas to GML Documents. *Journal of Information and Data Management*, 2(1):67--74.

[29] Jacobs, A. (2009). The pathologies of big data. *Communications of the ACM*, 52(8):36--44.

[30] Jagadish, H. V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M., Ramakrishnan, R., and Shahabi, C. (2014). Big data and its technical challenges. *Communications of the ACM*, 57(7):86--94.

[31] Kamel, I. and Faloutsos, C. (1994). Hilbert r-tree: An improved r-tree using fractals. In *Proceedings of VLDB.*, pages 500--509.

[32] Kim, G., Trimi, S., and Chung, J. (2014). Big-data applications in the government sector. *Commun. ACM*, 57(3):78--85.

[33] Le Pape, C., Gançarski, S., and Valduriez, P. (2004). Refresco: Improving query performance through freshness control in a database cluster. In *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, pages 174--193. Springer.

[34] Leavitt, N. (2010). Will NoSQL Databases Live Up to Their Promise? *IEEE Computer*, 43(2):12--14.

[35] Liu, C., Fruin, B. C., and Samet, H. (2013). SAC: semantic adaptive caching for spatial mobile applications. In *Proceedings of ACM SIGSPATIAL*, pages 174--183.

[36] Lu, Y., Cheng, J., Yan, D., and Wu, H. (2014). Large-Scale Distributed Graph Computing Systems: An Experimental Evaluation. *PVLDB*, 8(3):281--292.

[37] Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., Byers, A. H., and Institute, M. G. (2011). *Big data: The next frontier for innovation, competition, and productivity*. McKinsey Global Institute San Francisco.

[38] McAfee, A. and Brynjolfsson, E. (2012). Big data: the management revolution. *Harvard business review*, 90(October):60--68.

[39] Moro, M. M., Braganholo, V. P., Dorneles, C. F., Duarte, D., de Matos Galante, R., and dos Santos Mello, R. (2009). XML: some papers in a haystack. *SIGMOD Record*, 38(2):29–34.

[40] Nascimento, S. M., Tsuruda, R. M., Siqueira, T. L. L., Times, V. C., Ciferri, R. R., and de Aguiar Ciferri, C. D. (2011). The Spatial Star Schema Benchmark. In *Proceedings of GeoInfo*, pages 73–84, Campos do Jordão, Brazil.

[41] Niedermayer, J., Züfle, A., Emrich, T., Renz, M., Mamouliso, N., Chen, L., and Kriegel, H.-P. (2014). Probabilistic Nearest Neighbor Queries on Uncertain Moving Object Trajectories. *PVLDB*, 7(3):205–216.

[42] Poess, M. and Floyd, C. (2000). New TPC Benchmarks for Decision Support and Web Commerce. *ACM SIGMOD Record*, 29(4):64--71.

[43] Ray, S., Simion, B., and Brown, A. D. (2011). Jackpine: A Benchmark to Evaluate Spatial Database Performance. In *Proceedings of ICDE*, pages 1139--1150, Washington, DC, USA.

[44] Röhm, U., Böhm, K., and Schek, H.-J. (2000). OLAP query routing and physical design in a database cluster. In *Advances in Database Technology—EDBT 2000*, pages 254--268. Springer.

[45] Schiller, J. and Voisard, A. (2004). *Location-Based Services*. Morgan Kauffman, San Francisco, USA.

[46] Sellis, T., Roussopoulos, N., and Faloutsos, C. (1987). The r+-tree: A dynamic index for multi-dimensional objects. In *Proceedings of VLDB.*, pages 507--518.

[47] Shekhar, S. and Chawla, S. (2003). *Spatial Databases: A Tour*. Prentice-Hall, Englewood Cliffs, NJ, USA.

[48] Shekhar, S., Evans, M. R., Gunturi, V., Yang, K., and Cugler, D. C. (2014). Benchmarking Spatial Big Data. In *Specifying Big Data Benchmarks*, pages 81--93. Springer.

[49] Sidlauskas, D. and Jensen, C. S. (2014). Spatial Joins in Main Memory: Implementation Matters! *PVLDB*, 8(1):97--100.

[50] Simmen, D. E., Schnaitter, K., Davis, J., He, Y., Lohariwala, S., Mysore, A., Shenoi, V., Tan, M., and Xiao, Y. (2014). Large-Scale Graph Analytics in Aster 6: Bringing Context to Big Data Discovery. *PVLDB*, 7(13):1405--1416.

[51] Sowell, B., Salles, M. V., Cao, T., Demers, A., and Gehrke, J. (2013). An experimental analysis of iterated spatial joins in main memory. *PVLDB*, 6(14):1882--1893.

[52] Tahraoui, M. A., Pinel-Sauvagnat, K., Laitang, C., Boughanem, M., Kheddouci, H., and Ning, L. (2013). A survey on tree matching and XML retrieval. *Computer Science Review*, 8:1--23.

[53] Wright, A. (2014). Big data meets big science. *Commun. ACM*, 57(7):13--15.

[54] Wu, X. and Theodoratos, D. (2013). A survey on XML streaming evaluation techniques. *VLDB J.*, 22(2):177--202.

Appendix A

Detailed Experimental Evaluation

Here we show detailed information regarding the experiment evaluation (Section 5.4), these plots contain the same data analyzed previously. For each DBMS comparison in the performance evaluation, there is a detailed plot regarding the vertices per second distribution for each percentile interval. This visualizations help to understand how each DBMS behaves and which are its limits, whilst also comparing the performance between multiple systems. As in the previous plots, we consider data between the 5th and 95th percentiles, as well as *vertices per second* as our performance indicator.

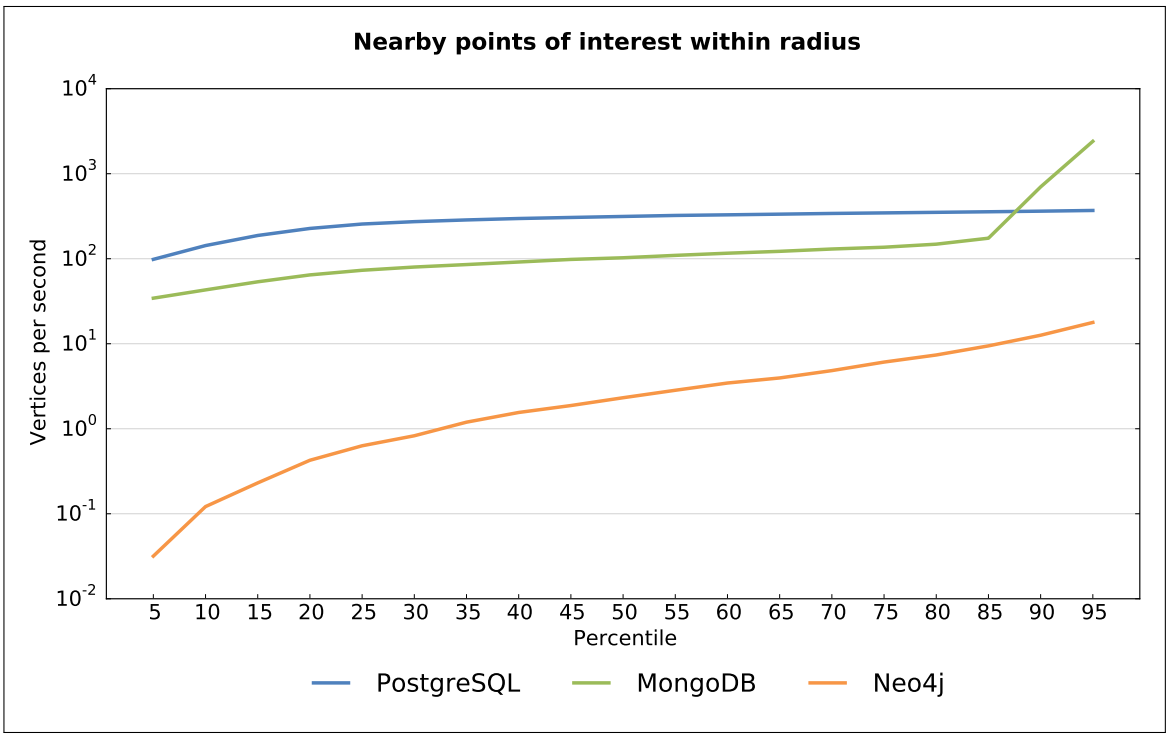


Figure A.1: Results for nearby POI within radius, percentile distribution.

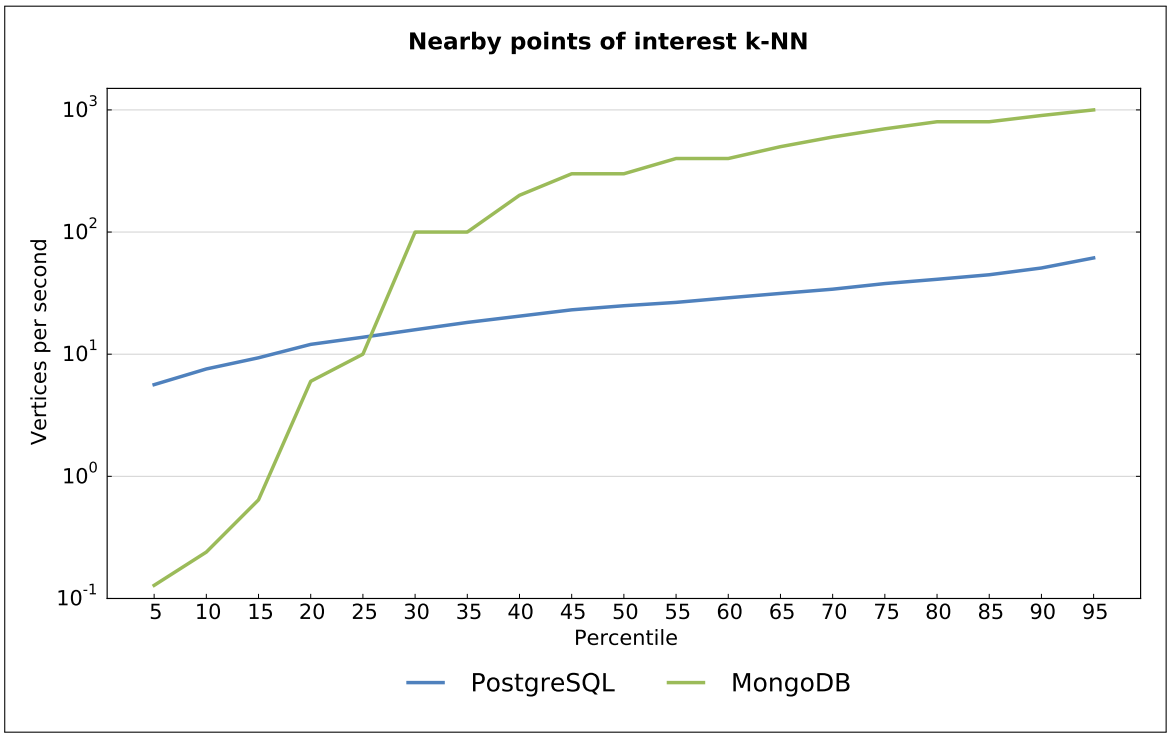


Figure A.2: Results for nearby POI k-NN, percentile distribution.

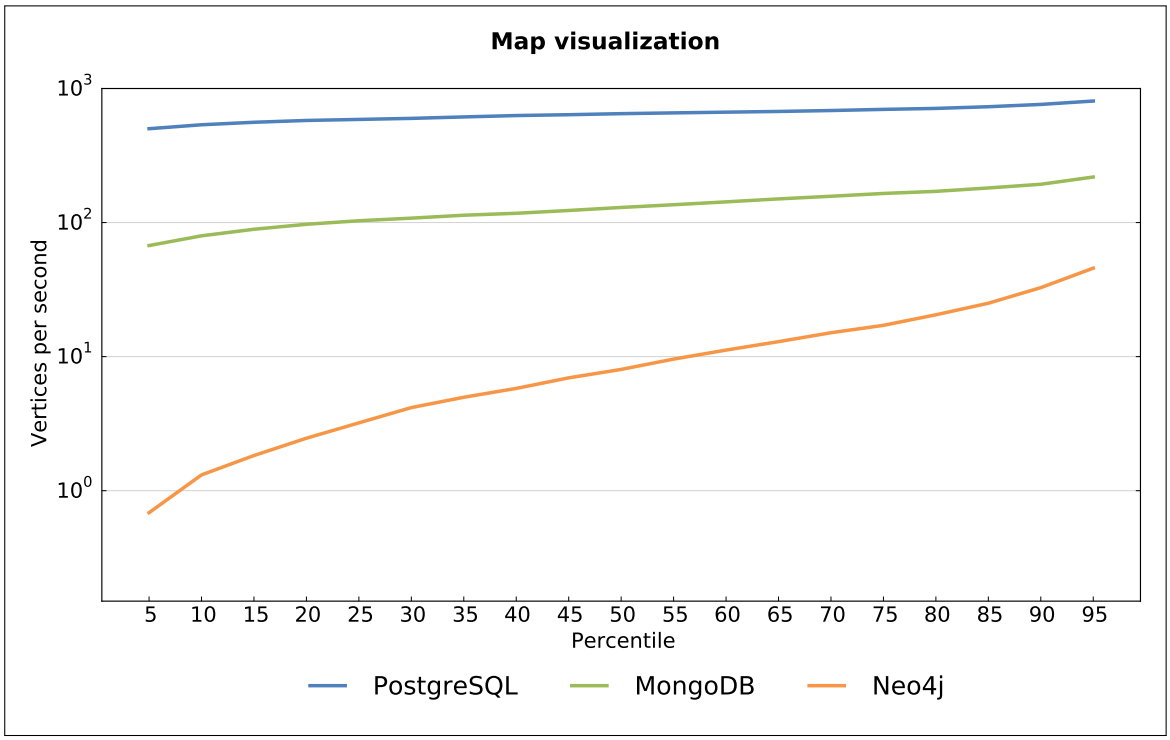


Figure A.3: Results for map visualization, percentile distribution.

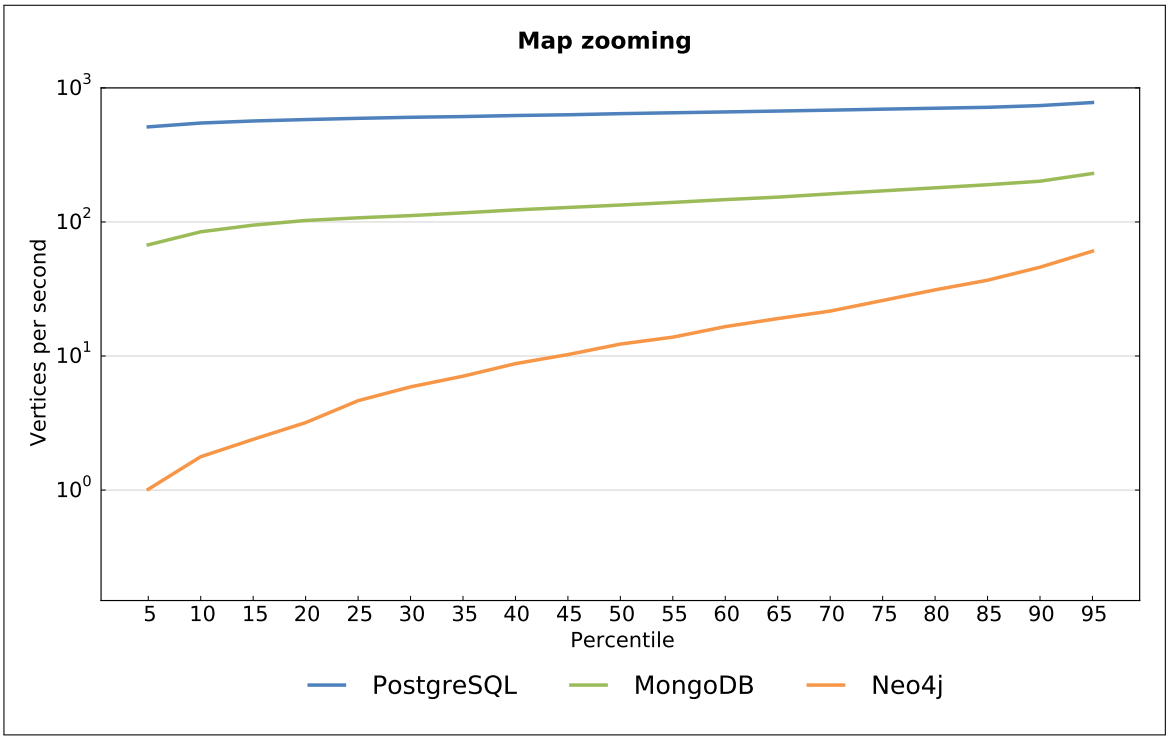


Figure A.4: Results for map zooming, percentile distribution.

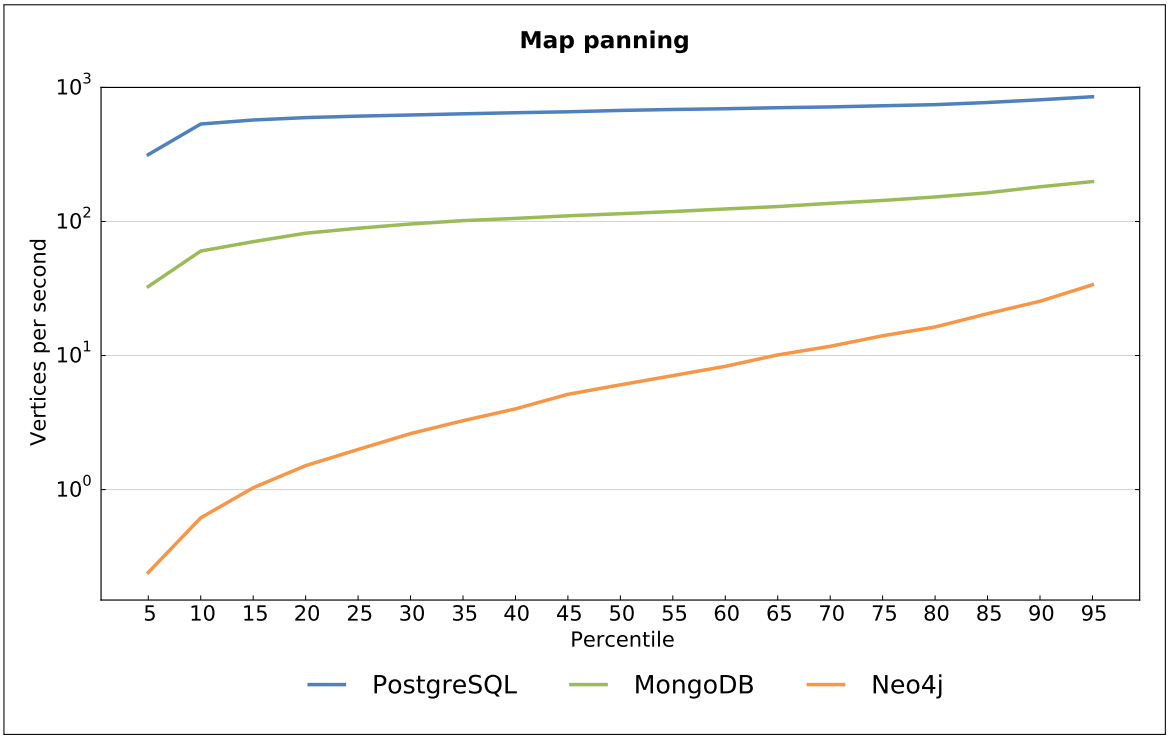


Figure A.5: Results for map panning, percentile distribution.

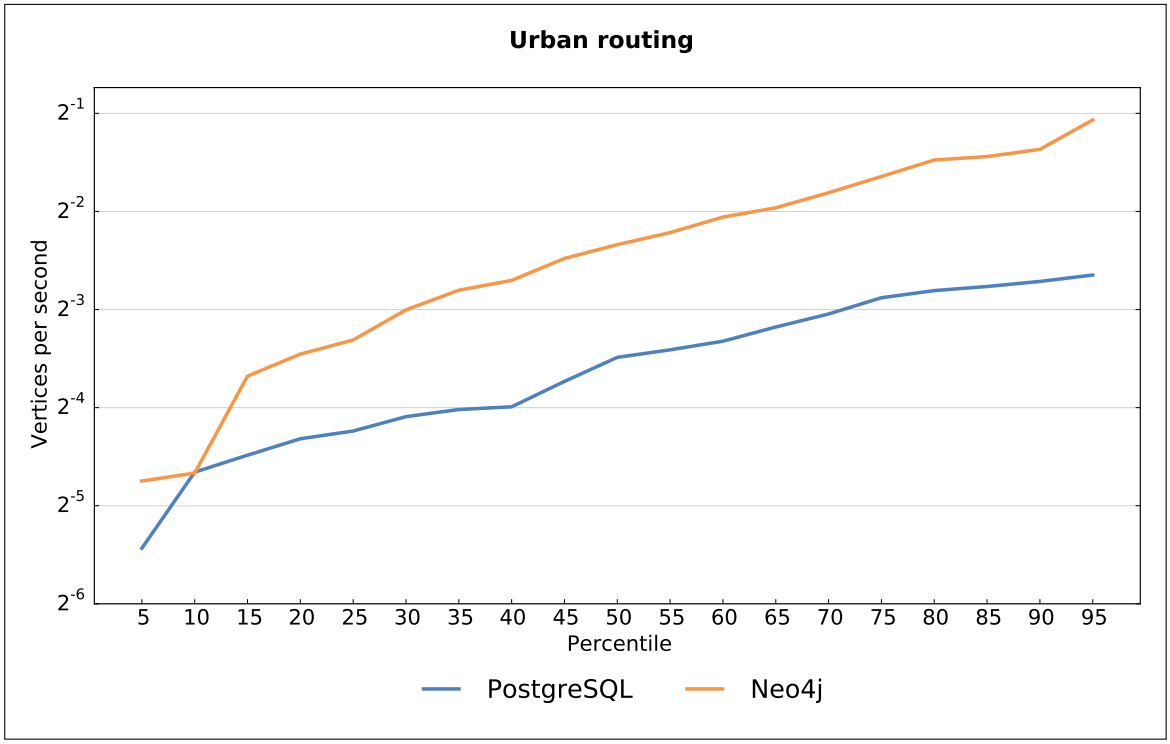


Figure A.6: Results for urban routing, percentile distribution.

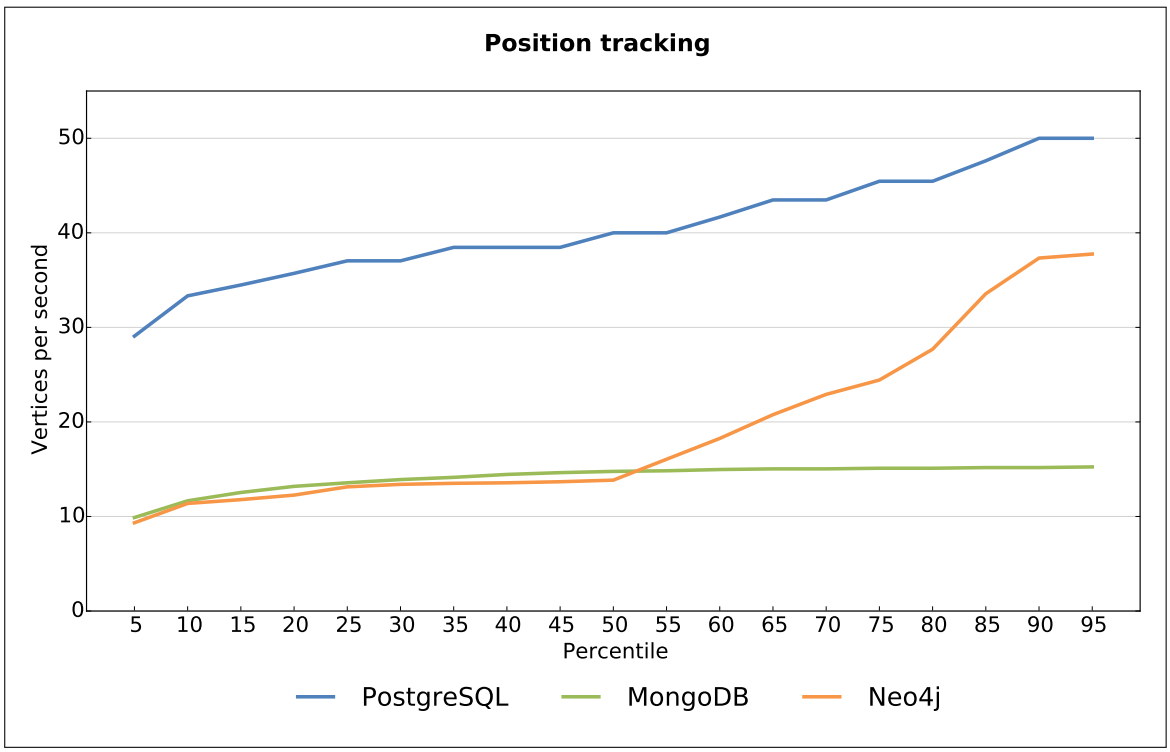


Figure A.7: Results for position tracking, percentile distribution.