ISTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

EXPLORING THE POSSIBILITIES OF GEOSPATIAL BIG DATA MANIPULATION USING NOSQL

M.Sc. THESIS

Ezgi ERGİN

Department of Applied Informatics

Geographical Information Technologies Programme

Thesis Advisor: Assoc. Prof. Dr. Ahmet Özgür DOĞRU

JUNE 2019

ISTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

EXPLORING THE POSSIBILITIES OF GEOSPATIAL BIG DATA MANIPULATION USING NOSQL

M.Sc. THESIS

Ezgi ERGİN
(706161005)

Department of Applied Informatics

Geographical Information Technologies Programme

Thesis Advisor: Assoc. Prof. Dr. Ahmet Özgür DOĞRU

JUNE 2019

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

NOSQL KULLANARAK MEKANSAL BÜYÜK VERİ İŞLEME OLANAKLARININ ARAŞTIRILMASI

YÜKSEK LİSANS TEZİ

Ezgi ERGİN
(706161005)

Bilişim Uygulamaları Anabilim Dalı

Coğrafi Bilgi Teknolojileri Programı

Tez Danışmanı: Assoc. Prof. Dr. Ahmet Özgür DOĞRU

HAZİRAN 2019

Ezgi Ergin, a M.Sc. student of ITU Informatics Institute student ID 706161005 successfully defended the thesis/dissertation entitled "EXPLORING THE POSSIBILITIES OF GEOSPATIAL BIG DATA MANIPULATION USING NOSQL", which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor : **Assoc. Prof. Dr. Ahmet Özgür DOĞRU**
İstanbul Technical University

Jury Members :

Date of Submission : 3 May 2019
Date of Defense : 14 June 2019

To my family,

FOREWORD

This thesis has been conducted with a collaboration of the Department of Geographical Information Technologies in Istanbul Technical University, Turkey and Geoinformatics Engineering Department in Politecnico di Milano, Italy, from February 2018 till May 2019.

During this long period in different countries, many people assisted me with great value, directly or indirectly. I would like to take this chance to thank all of them, starting with my advisor Assoc. Prof. Dr. Ahmet Özgür Doğru for all his understanding and support all the time for everything, without him I would not be able to accomplish this study and progress in my professional carreer. Besides, I would like to express my gratitude to Prof. Maria Antonia Brovelli and Daniele Oxoli for their help to structure the methodology of the study, seeking answers to all technical questions along with me and for their hospitality during my time in Italy; to all my colleagues in TomTom particularly Ravi Chauhan for providing me the valuable data I need and allowing me to study along with work; to my little brother Barış Ergin for his organizational support; to Ahmet Erdem for providing me the hardware and Candan Eylül Kilsedar for encouraging me all the time. And finally, last but not the least, my dear friend Merve Keskin for literally being by my side during the toughest time of this study and being a great companion.

Thank you all, I do appreciate all the help I received and I am grateful for everything I have learned during this study.

May 2019

Ezgi ERGİN
(Geomatics Engineer)

TABLE OF CONTENTS

Page

ABBREVIATIONS

ARPA	**:** Regional Agency for Protecting the Environment
GUI	**:** Graphical User Interface
ICT	**:** Information Communication Technology
IDE	**:** Integrated Development Environment
IoT	**:** Internet of Things
JSON	**:** JavaScript Object Notation
NoSQL	**:** Not Only Structured Query Language
SQL	**:** Structured Querying Language
WGS84	**:** World Geodetic System 1984

LIST OF TABLES

Page

LIST OF FIGURES

Page

EXPLORING THE POSSIBILITIES OF GEOSPATIAL BIG DATA MANIPULATION USING NOSQL

SUMMARY

With the rapid population increase in the cities, the need for a powerful, dynamic city management is becoming more crucial. In order to overcome the problems appearing as the cities grow (e.g. transportation, resource management, pollution, waste disposal, ect.), the smart city concept comes into prominence. Smart city is a comprehensive system that utilizes various data coming from different sources, provides storing, monitoring and analyzing infrastructure, and delivers solutions for the problems and activities. The handling of the unstructured data continuously coming from different sources and expanding in size (i.e. big data) is the main struggle. Especially, considering that most of the data is georeferenced, geospatial indexing and processing for that kind of big data is of high importance. Current traditional relational database systems have strong geospatial functionalities in a part of them (e.g. Postgresql), however they have two major limitations. First is the fixed column-based schema structure, which obstructs the unstructured data import, and the second is the hardware dependency on the performance. Therefore, to cover the scalability and flexibility needs of big data management, NoSQL (Not Only SQL) databases are developed. NoSQL databases are non-relational and unstructured, hence can store different types of data altogether. They are horizontally scalable, which means performance can be increased by adding machines into the system. Furthermore, NoSQL data storing functionalities support nested hierarchical data models, which is not available in relational databases. Nevertheless, it is still a discussion if SQL based relational databases can be replaced by NoSQL. Based on the studies comparing these two, not all NoSQL databases perform better than SQL databases. On the other hand, two of the most popular NoSQL databases (i.e. MongoDB and Couchbase) are found to have superior performance than corresponding SQL databases in many aspects. In this study, we aimed to experiment on MongoDB on a single server, by importing a big geolocated traffic data and pollution sensor data, and performing aggregate queries, geospatial functions and correlation analysis on it. Based on the results MongoDB showed a satisfying performance despite few constraints and gaps, especially for applying geospatial joins. This study can be carried further investigating the ways to execute more complex geospatial queryies on MongoDB, and by using larger data sets, additional servers, other NoSQL based systems and/or supportive tools.

NOSQL KULLANARAK MEKANSAL BÜYÜK VERİ İŞLEME OLANAKLARININ ARAŞTIRILMASI

ÖZET

Şehirlerdeki hızlı nüfus artışı ile birlikte güçlü, dinamik bir şehir yönetimine duyulan ihtiyaç gittikçe daha büyük bir önem kazanmaktadır. Şehirler büyüdükçe ortaya çıkan sorunların üstesinden gelmek için (ulaşım, kaynak yönetimi, kirlilik, atık bertarafı, vb.) akıllı şehir konsepti öne çıkmaktadır. Akıllı şehir, çeşitli kaynaklardan gelen farklı tipte verileri kullanan, depolama, gözlemleme ve analiz altyapısı sağlayan ve sorunlara ve aktivitelere yönelik çözümler sunan kapsamlı bir sistemdir. Bu tür bir sistemin en önemli girdisi değişik kaynaklardan sürekli olarak gelen ve biriken verilerdir.

Farklı veri kaynaklarından ve sensörlerden gelen kesintisiz veri akışı büyük boyutlu ve yapılandırılmamış bir veri havuzu oluşturur. Bu tür veriyi tanımlamak için büyük veri terimi ortaya çıkmıştır. Çeşitli uzmanlık alanları arasında farklı tanımları olmasına rağmen, büyük veriler için en yaygın kullanılan tanımı, geleneksel yöntemlerle kolayca depolanamayan, işlenemeyen veya analiz edilemeyen, çoğunlukla yapılandırılmamış büyük miktarlarda veridir. Büyük verilerin ana özellikleri, hacmi, kendi içinde çeşitliliği, farklı veri türleri ve yaklaşımlarına sahip olması, sürekli bir akış içinde birikmesi ve ortaya çıkarması zor olan yüksek değerler içermesidir. Şehirlerdeki çoklu veri kaynaklarından ve sensörlerden gelen büyük veriler trendler, davranışlar, gerçek zamanlı çözümler vb. hakkında değerli bilgiler içerir, bu sebeple analiz edilmeleri önemli bir konudur.

Büyük verilerle ilgili en önemli komplikasyon, verinin işlenmesi ve anlamlı bilgilerin çıkarılması olmuştur. Mevcut geleneksel yaklaşımda, SQL (Structured Querying Language) tabanlı ilişkisel veritabanları yaygın olarak kullanılır. SQL, karmaşık sorgular gerçekleştirme ve sağlam ve istikrarlı bir altyapı oluşturma kapasitesine sahip, güçlü bir sorgulama dilidir. Öte yandan, ilişkisel veritabanlarında veriler önceden oluşturulmuş veritabanı şemasına göre yapılandırılmalıdır, bu durum düzensiz verinin içe aktarımını engellemekte. çeşitli veri türlerini aynı yerde depolanmasını destekleyememekte ve faklı türde verileri ön düzenleme yapmadan birlikte işleyememektedir. Ayrıca performansı donanıma bağlıdır ve bilgisayar kapasitesi güçlendirilerek artırılabilir, ancak yine de donanım özellikleriyle sınırlıdır.

Geleneksel SQL veritabanlarının bu tür yetersizlikleri nedeniyle NoSQL (Not Only SQL) veritabanları geliştirilmiştir. Bunlar, geleneksel veritabanı sistemlerinin eksiklikleri olan esneklik ve performans sorunlarını çözümleyecek şekilde tasarlanmıştır. NoSQL veritabanları SQL tabanlı olanlar gibi standart bir sisteme sahip değildir. Her veri tabanı kendi modeline, veri formatına ve sorgulama diline sahiptir ve hepsi farklı amaçlara hizmet edebilir.

NoSQL, temel olarak geleneksel tablolar yerine yoğunluklu olarak “anahtar: değer” çiftleri ya da JSON (JavaScript Object Notation) benzeri döküman formatı kullanan, ilişkisel olmayan, düzensiz bir veritabanı sistemidir. Bu, yapılandırılmamış verilerin aynı veri tabanında depolanmasını ve elastik şema yönetimi ile esneklik sağlamaktadır. Performans donanıma daha az bağımlıdır ve yükü dağıtmak için daha fazla sunucu ya da bilgisayar eklenerek performans arttırılabilir, böylece daha büyük veri kümelerini yönetmek mümkündür. JSON benzeri veri formatı sayesinde NoSQL sistemler ilişkisel veritabanı sistemleri tarafından doğru şekilde desteklenmeyen ağaç benzeri hiyerarşik veri depolamayı destekleyebilimektedir. Benzer şekilde, yine klasik ilişkisel

veritabanı sistemlerinde uygulanamayan hiyerarşik veri modellerini de desteklenebilmektedir.
Bunlarla birlikte, NoSQL sorgulama dilleri standartlaştırılmamıştır, her veritabanının kendi sorgulama dili vardır, dolayısıyla karmaşık sorguları gerçekleştirmek SQL'e göre daha zorludur.
Ayrıca, yüksek frekanslı işlemleri desteklemek için kullanılabilse de, NoSQL veritabanları ilişkisel olanlar kadar stabil değildir.
Varolan verilerin çoğunluğunun coğrafi olduğu ve coğrafi referanslı verilerin miktarı her yıl ciddi bir şekilde arttığı düşünüldüğünde, bu tür büyük verilerin coğrafi olarak indekslenebilmesi ve işlenebilmesi büyük önem taşımaktadır. Bu sebeple SQL ya da NoSQL, coğrafi özellikli büyük veri işlenecek her veritabanında ölçeklenebilir coğrafi sorgulama ve analiz işlevlerinin olması ve yeterliliği mühim bir kriterdir.
Genel olarak SQL ve NoSQL tabanlı veritabanlarını karşılaştıran çalışmalara bakıldığında, NoSQL veritabanları her zaman SQL veritabanlarından daha iyi performans göstermemektedir. Öte yandan, en popüler NoSQL veritabanlarından ikisinin (MongoDB ve Couchbase), birçok açıdan karşılaştırıldıkları SQL veritabanlarından daha üstün performans gösterdiği gözlemlenmiştir. Özel olarak coğrafi sorgulama kapasitelerine bakıldığında ilişkisel veritabanlarının güçlü ve standart SQL tabanlı sorgulama özellikleri dolayısıyla oldukça komplike coğrafi analizler yapabildiği, NoSQL veritabanlarının ise bir kısmının coğrafi indexleme kapasitesiyle beraber temel coğrafi analizleri de yapabildiği ve üçüncü parti yazılımlarla daha da derin analizler koşturulabildiği görülmektedir.
Bu çalışmada, coğrafi büyük verileri bir NoSQL veritabanına aktarma olasılıklarını ve metodolojilerini araştırmak ve bu veri tabanının verimliliğini veri işleme, sorgulama, coğrafi fonksiyonlar ve analizler açısından test etmek amaçlanmıştır. Uygulama için Milan şehrinin bir senelik büyük trafik ve hava kirliliği verileri NoSQL veritabanına işlenmiş, ayrıca bu veritabanı tarafından desteklenen bazı temel mekansal işlevler test edilmiştir. Kullanılan trafik verisi, trafikteki araçlardan yaklaşık 30 saniyelik frekansla alınan hız verisidir. Tekil araç bilgisi içerdiği için hassas ve gizli olan bu veri anonimize edilerek kullanılmıştır. Hava kirliliği verisi ise Milan şehri merkezini kapsayan 11 hava gözlem istasyonunda toplanan ve internet üzerinden paylaşılan gözlemlerden alınmıştır.
Araştırma yaygın kullanılan, JSON benzeri döküman tabanlı ve ücretsiz bir NoSQL veritabanı olan MongoDB için yapılmıştır. Bu veritabanının tercih edilmesi mekansal indeksleme seçeneğine sahip, büyük veri erişiminde ve sorgulamalarında verimli, esnek ve yaygın kullanımı nedeniyle önemli miktarda dokümantasyona sahip olmasından ötürüdür.
Çalışma kapsamında eldeki veri kullanım koşullarına uygun bir şekilde işlenerek MongoDB içine aktarılmış ve temel toplu (aggregate) sorgular, coğrafi fonksiyonlar ve korelasyon analizleri denenerek veritabanı test edilmiştir. Verilerin hazırlanması ve aktarımı python betik dili ile yapılmıştır. Veritabanına erişim ve sorgulamalar için MongoDB Compass ve NoSQLBooster programlarından faydalanılmıştır.
Çalışmanın sonuçlarına göre MongoDB bazı veritabanının standart fonksiyonlarına bağlı sınırlamalar (örneğin korelasyon analizinin direk olarak mümkün olmaması) ve mekansal katman birleştirme konusundaki eksiklerinin dışında olumlu bir performans göstermiştir.
Bu bağlamda, toplu sorgular ve tek koleksiyonlardaki temel mekansal sorguların MongoDB'nin güçlü yönü olduğu görülmektedir. Fakat karmaşık sorgulama ve mekansal birleştirme fonksiyonlarında zayıflıkları bulunmaktadır. Genel olarak, üçüncü parti yazılımların daha karmaşık analizler yapmak için kullanılabileceği göz

önüne alındığında, MongoDB'nin büyük veri yönetimi için tercih edilebilir bir veritabanı olduğu söylenebilmektedir.
Bu çalışma MongoDB üzerinde karmaşık mekansal sorguların uygulanma yollarının araştırılmasıyla, daha büyük veri setleri ve ek sunucu kullanımı ile, diğer NoSQL tabanlı sistemler ve destekleyici araçların incelenmesiyle ve elde edilen analiz sonuçlarından şehir yönetimi konusunda faydalanılma olanaklarının tahkik edilmesiyle farklı açılardan daha ileri götürülebilmesi mümkündür.

1. INTRODUCTION

In this age, human population is fast moving from rural areas to cities. According to United Nations, by 2050, 70% of the population will be living in the urban areas (Diaconita et al., 2018). The drastic increase of the inhabitance in cities will also bring problems in urban planning, employment, habitation, transportation, share of energy sources, natural resources management, pollution and so forth. Hence, structuring a well-organized, flexible, scalable, efficient and sustainable city planning is very important in that aspect which brings us to smart city concept. Smart city is a digital infrastructure using Information Communication Technology (ICT) and Internet of Things (IoT) technologies, in other words, a thorough system providing storage, monitoring, analysis and solution of the activities and problems in cities. Continuous data collection from different sources in the urban zone such as networks, services, cameras and sensors is crucial for smart cities (Malik et al., 2017). Especially, sensor data, which is basically an output from any kind of sensor storing and/or reacting to changes in surroundings, can be considered as the main data source for IoT and smart cities. As it is continuously streamed, the size of this sensor data can be huge and the growth in the unstructured data obtained from different sources introduced the term big data that is one of the most important trending topics (Li et all, 2016).

Big data has long been a controversial topic due to the difficulties of describing its characteristics and therefore developing the methodology to extract meaningful information. Divisions on big data is mostly on the definition of it across different domains, still most common definition of big data is, huge amounts of data which is generally unstructured, cannot be easily stored, processed or analyzed with conventional methods (Li et all, 2016). In addition to that definition, Chen et al. (2014) reviewed several different definitions of big data and summarized the main features of it as 4Vs: Volume (i.e. big size data), Variety (i.e. different data types and approaches), Velocity (i.e. continuously populated and streamed), Value (contains high value that is hard to extract). However, the most significant complication about big data is not

the definition of it, but the processing and extraction of meaningful information (i.e. value) from it have been the hardest to tackle on big data.

In traditional approach, SQL (Structured Querying Language) based relational databases are commonly used with a very strong querying language to handle the data with capability to perform complex queries and establishing a robust and stable infrastructure. On the other hand, relational databases are table based with fixed columns and data should be structured according to the pre-constructed database schema, which is resulting in a rigid structure to be able to insert because it cannot support various data types. Performance depends on the hardware, and can be increased by upgrading it, yet still limited to hardware capabilities.

Due to these limitations, NoSQL (Not Only Structured Query Language) logic is developed. NoSQL is a non-relational, distributed database system using mainly ‘’key:value” pairs as documents instead of traditional tables. This approach allows storing unstructured data altogether in the same database, thus it provides flexibility with elastic schema management. Performance is less dependent on the hardware and can be increased by adding more servers to distribute the load, therefore it is possible to manage much larger datasets. Thanks to JSON (JavaScript Object Notation) like data format, NoSQL supports tree-like hierarchical data storage which is not properly supported by relational database systems. Hierarchical data model support is a significant reason to prefer NoSQL for big data processing. Along with these, NoSQL querying languages are not standardized, each database has its own querying language, consequently, performing complex queries is less efficient than SQL. Moreover, although it can be used as a database to support high amount of actions such as purchasing, NoSQL is not as stable as relational databases.

It is important to underline the fact that most of the data is georeferenced, and according to the common assumption 80% of it is spatial (Hahmann et al., 2011). Furthermore, as indicated by Lee and Kang (2015), the percentage of the geolocated data is drastically rising which is an evidence that personal location data amount is increasing by 20% every year. To handle such geospatial big data, the database, whether traditional SQL or new generation NoSQL should have scalable geospatial data processing features.

In this context, the aim of the study is to investigate the possibilities and methodologies to import big geospatial data (e.g. satellite imageries, mobile tracking, traffic, meteorology, temperature, pollution, etc.) into a NoSQL database and to test efficiency of that database in terms of data processing, querying, geospatial functions and analyzing. Although there are some studies (see Chapter 2) focusing on NoSQL database performance on web services, real time applications, comparisons vs classic relational database systems on architecture, there is still a little known about querying and geospatial analysis functionalities of NoSQL database systems. Therefore, we will use big traffic and pollution data to be processed and imported in one of the NoSQL databases; and test some basic geospatial functions that is supported by that database.

In the next chapter, we will present a literature overview on relational and non-relational databases, their applications, and geospatial functionalities and capabilities. In Chapter 3, a case study to test functionalities of a NoSQL database will be explained, and this chapter is followed by the results of the case study including the problems faced during the application. Finally, in Chapter 5, general conclusions will be drawn, and further possibilities will be discussed.

2. LITERATURE REVIEW

With the recent advances in technology, new data sources arise and the need for rapid information exchange is emerging. However, data collected from various sources in large amounts, which is called big data, brings its own challenges for storing, management and processing. Li et al. (2016) listed some of the fundamental challenges as efficient representation and modeling, analyzing, mining and visualizing and quality assessment of geospatial big data. According to them, further development and research needs to focus on the following areas: real time spatial indexing algorithms, better data mining algorithms, more efficient and complex visualization considering task and user needs (e.g. online 3D visualization tools), more effective quality assessment approaches, more sophisticated definition of semantics and ontology relationships.

To meet the most of the needs mentioned above, non-relational databases have been developed and rapidly replacing the relational databases such as PostgreSQL, Oracle and MySQL. Unlike relational databases, these non-relational databases which are called as NoSQL vary in terms of their architecture, flexibility, scalability and abilities to store, manage, query and transfer the data. Existing NoSQL data system architectures can be classified as key:value, document, graph and column based databases (Zafar et al., 2016). Figure 2.1 demonstrates the data model, strength and weaknesses of these four database types in detail by providing examples. Querying varies in each of these NoSQL database types and majority of them requires additional scripts to perform complex queries.

Along with the new approaches NoSQL provides, it still remains a question whether NoSQL databases can completely replace traditional relational databases. Therefore, comparisons between the performances of SQL and NoSQL databases are required in different aspects.

In 2013, Li and Manoharan tested the performances (i.e. run time) of basic operations which are instantiation, read, write, delete and iteration for six NoSQL databases (i.e.

Cassandra, Couchbase, CouchDB, Hypertable, MongoDB, RavenDB) and compared with Microsoft SQL Server Express. According to the results, instantiation for MS SQL Server, Couchbase and MongoDB is significantly slower than others. For read and delete operations, MongoDB and Couchbase performed better than MS SQL Server compared to the others, although for write operation Couchbase, MongoDB, Cassandra and Hypertable were better than MS SQL. In terms of iteration no valuable difference observed in the performance.

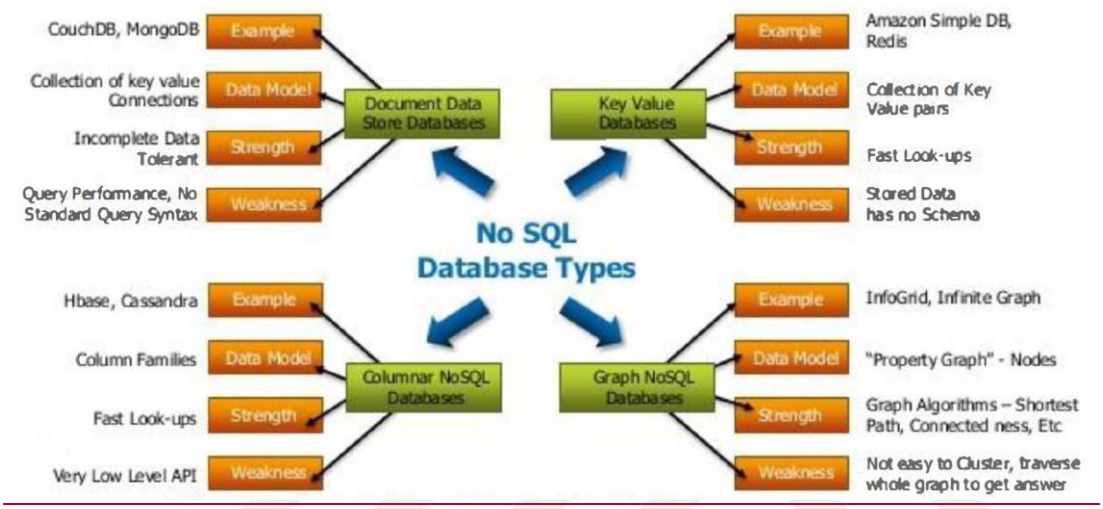


Figure 2.1 : Database Types (Zafar et al., 2016).

In another study, the functionalities of the most popular four databases reviewed together as seen in Table 2.1, and an experiment performed to compare the webservice response times for relational Azzure SQL database and NoSQL based Azzure Document DB (Baralis et al., 2017). Results revealed that Document DB responses considerably faster than Azzure SQL, whereas Azzure SQL is better in managing simultaneous requests.

A different SQL versus NoSQL comparison based on the insert time, disk usage, memory usage and querying time is executed by Lian et al. (2018) using MongoDB and PostgreSQL. MongoDB was found explicitly advantageous for insert time and querying time, and no remarkable difference observed for memory usage, nevertheless in terms of disk usage PostgreSQL was found to be more beneficial.

It has been mentioned in previous chapter that most of the big data is spatial, hence one of the most important aspects to evaluate is the geospatial functionalities of NoSQL. Although there are numerous NoSQL databases present in the market now, only few of them are spatial. Agarwal and Rajan (2017) underline that spatial

Table 2.1 : Qualitive comparison of the functionalities of four geospatial databases (Baralis et al., 2017).

Database	Supported Geometry objects	Main supported geometry functions	Supported Spatial indexes	Compatibility with GeoServer	DaaS	Horizontal scalability
PostGIS	Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeometryCollec tion	PostGIS supports the Open Geospatial Consortium (OGC) methods on geometry instances	B-Tree index R Tree index, GiST index	Yes	No	No
Azure SQL Database	Point LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeometryCollection	Azure SQL Database supports the Open Geospatial Consortium (OGC) methods on geometry instances	2d plane index, B-trees	Yes	Yes (Microsoft Azure cloud computing platform)	No
MongoDB	Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeometryCollection	Inclusion, Intersection, Dis tance/Proximity	2dsphere index, 2d index	Yes (based on the unsupported external MongoDB plug-in included in GeoTools)	Yes (MongoDB Atlas cloud service)	Yes (sharding)
DocumentDB	Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeometryCollection	Inclusion, Dis- tance/Proximity	2d plane index, quadtree	Yes (based on the unsupported external MongoDB plug-in included in GeoTools)	Yes (Microsoft Azure cloud computing platform)	Yes (sharding)

functionalities are quite recent for NoSQL databases, still a lot of improvement and investigation are needed. They also summarized the geometric operations of PostgreSQL/PostGIS, MongoDB and CouchDB as seen in Table 2.2, and tested the ones present in both PostgreSQL/PostGIS and MongoDB. As a result, MongoDB run almost 10 times faster for the common functions between them. This demonstrates that while PostgreSQL/PostGIS has wider geospatial functionalities, MongoDB performs faster with limited capabilities.

Table 2.2 : Geo-functions of the databases (Agarwal, S., & Rajan, K. S., 2017).

PostGIS	MongoDB	CouchBase
ST Within	$geoWithin	BBOX
ST Intersects	$geolntersects	
ST DWithin + Order by dist	$near + param(Distance)	
ST Area		

Zhang et al., (2014) has investigated the performance of storage and accessibility by importing and storing a big shapefile into MongoDB. According to them, MongoDB was considerably stronger than traditional relational database systems for handling massive amounts of data.

Although there is still a need for more research to investigate the superiorities and gaps of NoSQL databases, and to evaluate advance functionalities with recent developments, the studies carried out until today indicate that MongoDB and Couchbase performs better than SQL databases in most cases. MongoDB and Couchbase perform significantly better especially when the shared functionalities between relational and NoSQL databases are considered. However, the rest of the NoSQL databases have weak performances compared to both relational and MongoDB and Couchbase databases. For that reason, we decided to use MongoDB in our study by thoroughly investigating both the basic functionalities including import of a big data, aggregate and spatial queries, more complex analysis such as correlation.

3. CASE STUDY

The purpose of the case study is to evaluate implementation and processing of big geospatial traffic data and pollution data from sensors in a NoSQL database. For that, first all data is preprocessed to implement in the database, and after importing basic aggregate and geospatial queries performed. Additionally, correlation between traffic and pollution data is calculated.

3.1 Study Area

The study performed for the city of Milan, the capital of Lombardy region and second biggest city in terms of population in Italy. Specific area is selected within a bounding box around the city center, as seen in Figure 3.1.

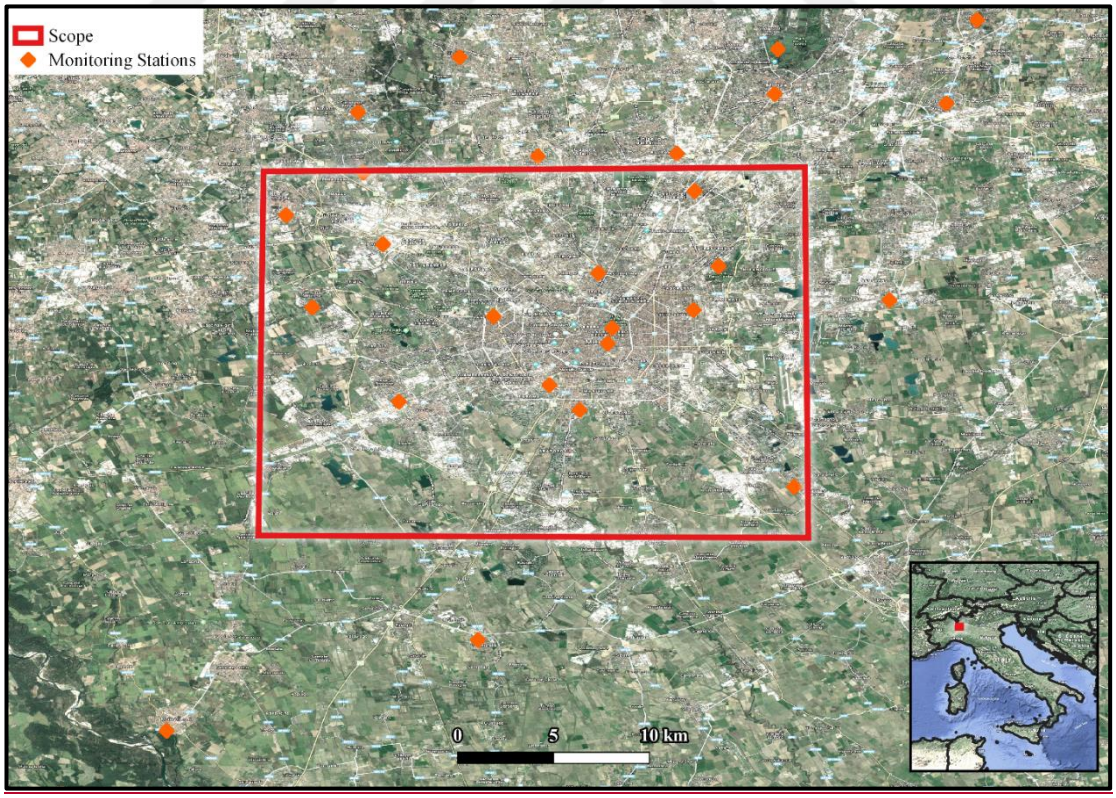


Figure 3.1 : Study area.

The date is limited to a range from 01/01/2016 to 31/12/2016 for a complete year, based on available verified data. Pollution data is obtained from ARPA Lombardia (Agenzia Regionale per la Protezione dell'Ambiente Lombardia, in English: Regional Agency of Lombardy for Protecting the Environment), and traffic data is attained from a European navigation, mapping and traffic company TomTom B.V.

3.2 Materials

3.2.1 Data

In this section, detailed information on characteristics, acquisition, distribution, temporal availability and completeness of the data used in the study is given, as this information is significant for interpreting results.

3.2.1.1 Traffic data

The data provided by TomTom B.V. Traffic Center is the probe counts product consisting of the speed data per vehicle with less than half minute temporal accuracy. This is a historical data collected from the vehicles in the traffic using TomTom products. From this data it is also possible to derive the number of cars present in the roads at a certain moment. As an important point, the provided data is of a specific TomTom product which stores data for a filtered set of vehicles. It contains only a part of the vehicles in the traffic, it does not have full coverage of the real-world situation and limited to the vehicles registered in this product. It is designed that way due to business reasons and to guarantee consistency within the product.

It is necessary to underline that this data is very sensitive and confidential since it has location and speed of each vehicle with a very high temporal intensity, therefore cannot be shared outside the company. Even within the company, the data for a single vehicle is not traced and never used alone. The products derived from this data are statistical information on traffic behavior, such as intensity of vehicles on a road per week days, business hours, day and night, ect. [Url-1]. For this reason, we had to anonymize the data before any use.

Similar to the area and date range defined for the case study, traffic information was gathered for the center of Milan, from 01/01/2016 to 31/12/2016. The huge amount of

data was delivered in four datasets, one for each tile covering the city center, as shown in figure 3.2. Datasets include textual information with millions of rows, ordered as slices per road edge. Normally raw vehicle probe data has point locations, but this product is processed and mapped to TomTom road elements and raw locations are redundant. Each slice in datasets contains information on road edge at first row and the rest belongs to the vehicle information. Among vehicle information, only epoch time and speed of the vehicle are considered for the study. The size of the four datasets for center of Milan in year of 2016 is around 111 GB in total. Additionally, as these datasets do not contain the geometries of roads, geometry is obtained from TomTom 2016 base map.

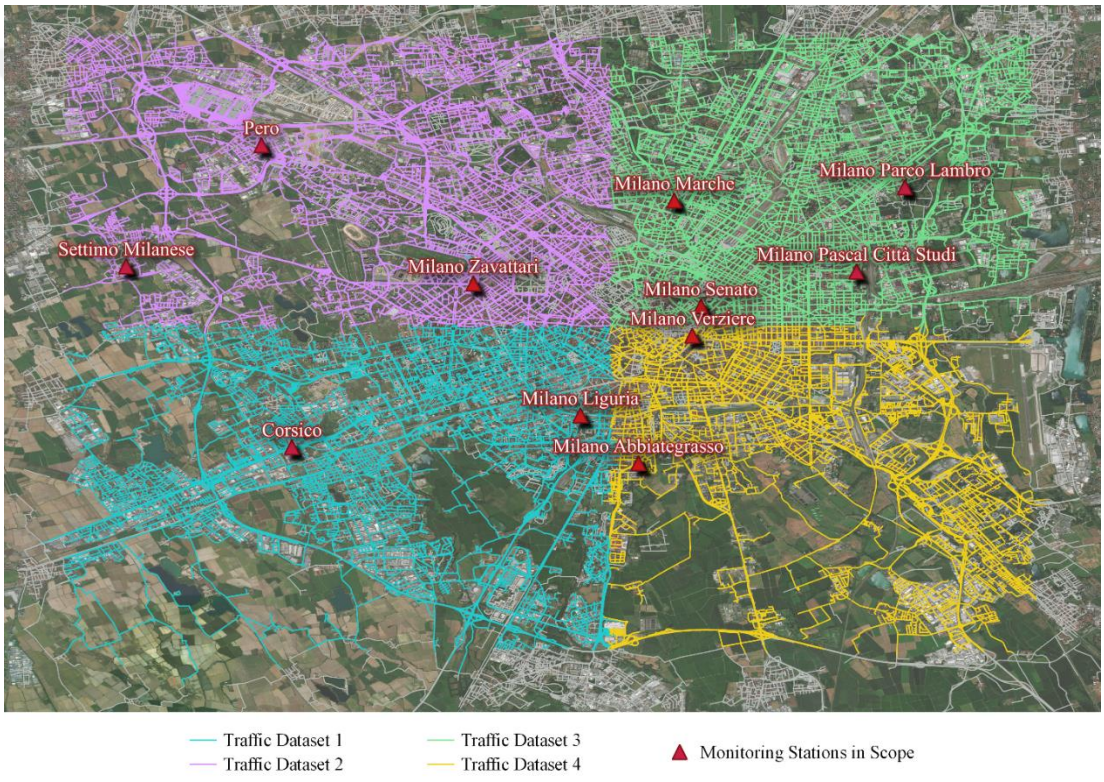


Figure 3.2 : Distribution of the data.

3.2.1.2 Pollution data

Pollution data for the city of Milan is retrieved from the observations published by the ARPA Lombardia, which is the agency dealing with the environmental issues of Lombardy region in Italy. Their activities contain monitoring the environmental indicators such as water, air, waste, soil, natural hazards, noise etc., and taking preventive and actions [Url-2]. In this study, we focus on the observations related to

air pollution. ARPA Lombardia has total 137 air quality monitoring stations installed in Lombardy, stores the pollutant values such as Nitrogen oxides (NO/NO2), Particulate Matter (PM10/PM2.5), Carbon monoxide (CO), Black Carbon (BC), Benzene (C6H6), Sulfur dioxide (SO2) with hourly frequency. The observed values are released in the official website of ARPA Lombardia, besides, validated by the agency except for the data from last six months.

For this study, hourly observations from 11 stations in center of Milan are considered. Distribution of the stations in scope can be seen in figure 3.2. The data is downloaded with a date range from 1/1/2016 to 31/12/2016. The reason to use the data from 2016 is to benefit from the most recent officially validated pollution data by ARPA Lombardia at the time the study started.

Another point to mention is that there are some gaps in the data, values are not available for every hour. Furthermore, not all pollutants are observed in all stations, each station monitors a limited set of pollutants. In Table 3.1 the completeness of the pollutants per station can be seen.

Table 3.1 : Availability percentages of the pollution data.

Station name	Pollutant	Formula	availability
Milano - viale Marche	Nitrogen dioxide	NO2	98.7%
Milano - viale Marche	Carbon monoxide	CO	97.6%
Milano - viale Marche	Nitrogen oxides	NO	98.7%
Milano - viale Marche	Benzene	C6H6	91.4%
Milano - P.zza Zavattari	Nitrogen dioxide	NO2	96.2%
Milano - P.zza Zavattari	Carbon monoxide	CO	94.5%
Milano - P.zza Zavattari	Benzene	C6H6	84.7%
Milano - P.zza Zavattari	Nitrogen oxides	NO	96.2%
Milano - Verziere	Nitrogen dioxide	NO2	98.5%
Milano - Verziere	Ozone	O3	90.0%
Milano - Verziere	Nitrogen oxides	NO	98.4%
Milano - Verziere	Particulate matter	PM10	96.4%
Milano - viale Liguria	Nitrogen dioxide	NO2	88.3%
Milano - viale Liguria	Carbon monoxide	CO	92.5%
Milano - viale Liguria	Nitrogen oxides	NO	88.4%
Milano - Parco Lambro	Nitrogen dioxide	NO2	95.3%
Milano - Parco Lambro	Ozone	O3	89.9%
Milano - Parco Lambro	Nitrogen oxides	NO	95.3%

Table 3.1 (continued) : Availability percentages of the pollution data.

Station name	Pollutant	Formula	availability
Milano - via Senato	Nitrogen dioxide	NO2	97.0%
Milano - via Senato	Benzene	C6H6	85.0%
Milano - via Senato	Carbon monoxide	CO	91.0%
Milano - via Senato	Nitrogen oxides	NO	97.0%
Milano - via Senato	BlackCarbon	BC	87.1%
Milano - via Senato	Particulate matter	PM10	96.4%
Milano - via Senato	Particulate matter	PM2.5	93.7%
Milano - P.zza Abbiategrasso	Nitrogen dioxide	NO2	88.7%
Milano - P.zza Abbiategrasso	Nitrogen oxides	NO	88.6%
Milano - Pascal Città Studi	Benzene	C6H6	91.9%
Milano - Pascal Città Studi	Ammonia	NH3	84.7%
Milano - Pascal Città Studi	Nitrogen dioxide	NO2	88.6%
Milano - Pascal Città Studi	Sulfur dioxide	SO2	87.5%
Milano - Pascal Città Studi	BlackCarbon	BC	99.2%
Milano - Pascal Città Studi	Particulate matter	PM2.5	89.9%
Milano - Pascal Città Studi	Nitrogen oxides	NO	90.0%
Milano - Pascal Città Studi	Ozone	O3	98.8%
Settimo Milanese	Nitrogen dioxide	NO2	81.9%
Settimo Milanese	Nitrogen oxides	NO	81.9%
Corsico	Nitrogen dioxide	NO2	96.2%
Corsico	Ozone	O3	94.1%
Corsico	Carbon monoxide	CO	87.4%
Corsico	Nitrogen oxides	NO	96.2%
Pero	Nitrogen dioxide	NO2	83.6%
Pero	Nitrogen oxides	NO	83.3%

3.2.2 Software used in the study

Based on the literature review, MongoDB is recognized to be the most convenient NoSQL database for this study due to its geospatial functionalities and processing performance, therefore application is done with that database.

MongoDB is a free to use, JSON like document based NoSQL database used by many big companies which is highly flexible, efficient in accessing and querying big data, having also spatial indexing option [Url-3]. As it is one of the most widely used NoSQL databases, there is a considerable amount of documentation and information exchange on any kind of issues related to MongoDB.

MongoDB is structured with collections under databases, which are bundles of documents. Documents are the single entries of a database. There is also a term cluster in MongoDB terminology which is the pieces of server called shrad that forms the collections and databases, which shows the horizontal scalability of MongoDB, as seen in figure 3.3.

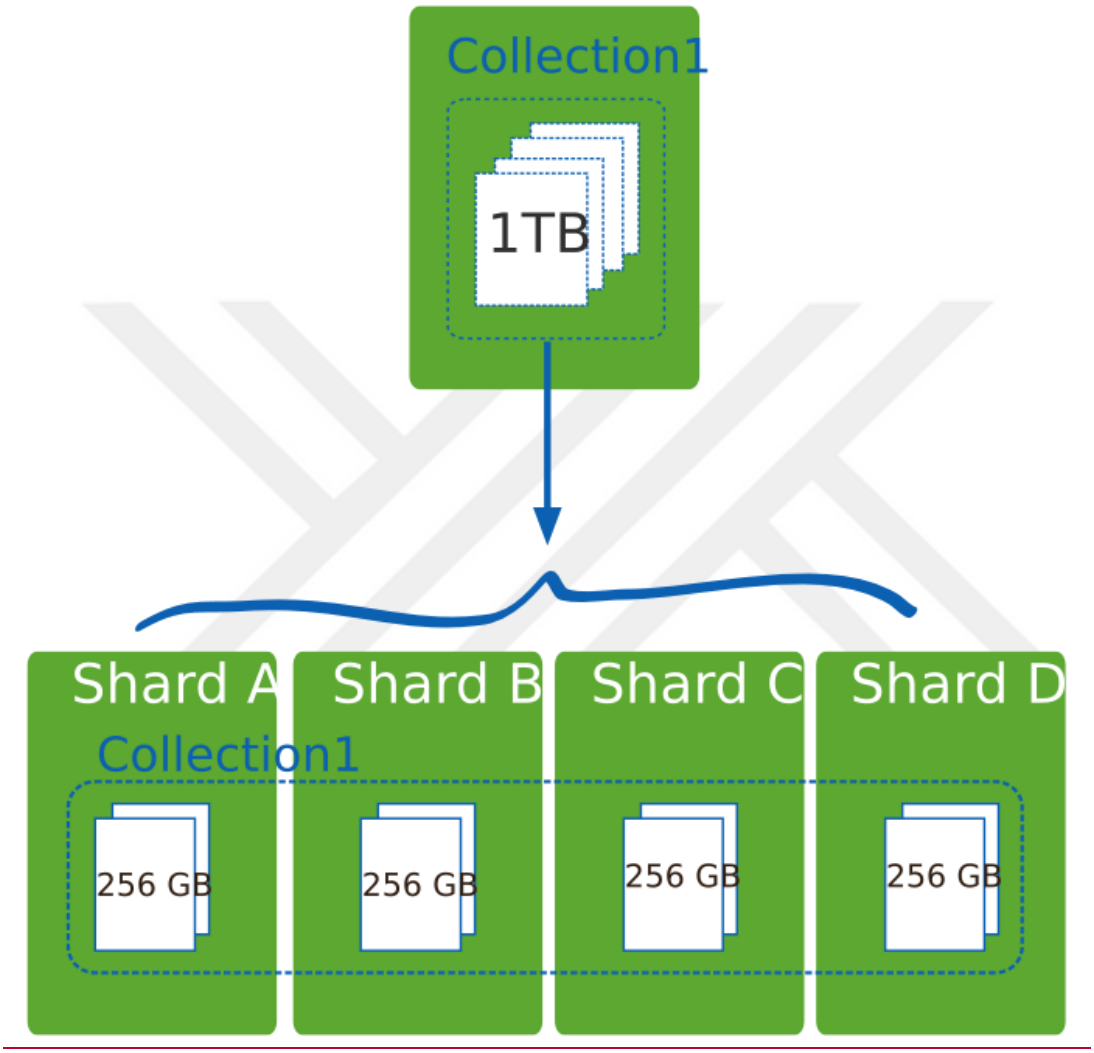


Figure 3.3 : Shrads distributes data over multiple servers [Url-4].

For database management MongoDB Compass is used, which is the GUI (Graphical User Interface) coming with MongoDB installation. Preprocessing of data and imports into Mongodb are done using Python scripting language. Additionally, an open source software NoSQLBooster, which is an IDE (Integrated Development Environment) for MongoDB used for building and running queries on the database [Url-5].

3.2.3 Hardware used in the study

The computer used in the study has below system properties:

- Disk: 250GB SSD (Solid State Drive)
- RAM: 8GB (+8GB Virtual RAM)
- Processor: Intel Core i7-2630QM CPU@ 2.00GHz x 8
- Operating system: Linux Ubuntu 18.04 (bionic)

3.3 Methodology

The methodology applied in this study is shown in the flowchart in Figure 3.4, and all steps after data acquisition are explained below.

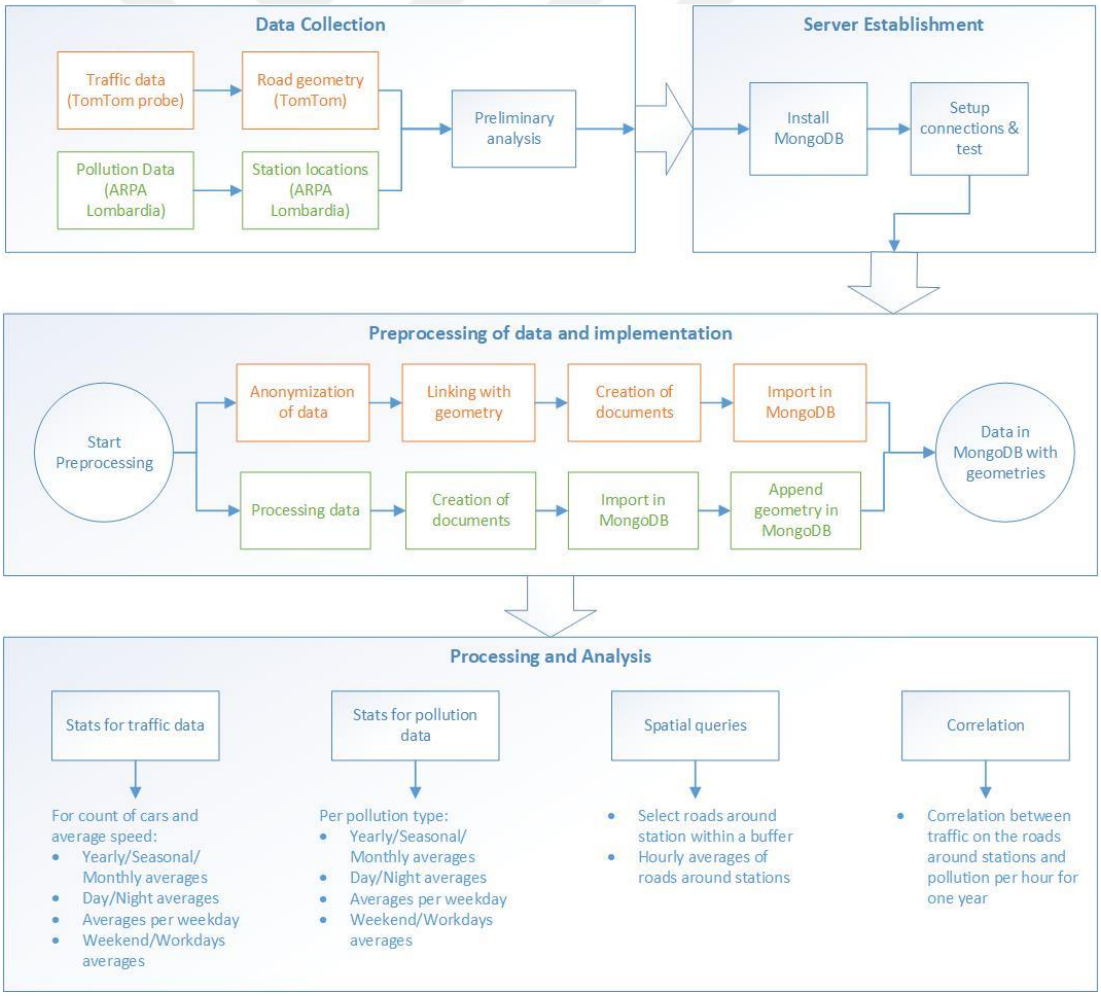


Figure 3.4 : Workflow.

3.3.1 Server establishment

As database, MongoDB 4.0.9 Community version was setup. Installation included the MongoDB GUI for viewing and managing the database. Server was established locally on desktop within one cluster, not sharded. One database was created (named as "milan") to put all the data in, and inside this database, two collections were formed; "probe" and "pollution".

To preprocess the data into document format and import it into the database, Python 3.6 with Anaconda distribution was installed due its effective library handling capabilities. For preprocessing and database connections, pandas, numpy and pymongo libraries are also utilized. Lastly, NoSQLBooster 5.7.1 was installed to perform queries on the database.

3.3.2 Preprocessing of the data

MongoDB is a document based database as mentioned in previous chapters, thus, the main preprocessing step is to format all the data as documents to be able to import into database.

3.3.2.1 Traffic data preprocessing

The traffic data consists of slices of information per road edge, but as the count of vehicles per road is not known, the beginning and the ending of these slices are not known as well. As the first step, a list of beginning and ending line numbers were extracted from the datasets, by reading through all file and detecting keywords for the beginning of slices. This list of line numbers was used as an input for the rest of preprocessing.

As mentioned in Chapter 3.2.1.1, the traffic data is confidential and sensitive, and before using it must be anonymized. For practical reasons, this anonymization was done by aggregating and calculating the average speed and total amount of vehicles per hour for each road and excluding the rest of the information. In this way, traffic data and pollution data have become consistent as both have hourly frequency. Another important reason for this decision is, the 16MB document size limitation in MongoDB [Url-6].

Using the list of line numbers, all slices per road were detected and aggregated per hour. Aggregation output was translated into a document format containing road id (e.g. segment id), direction of flow, total amount of vehicles and a list of values. This list contains time, average speed and count of vehicles for each hour where data is present. Following this, the geometry from 2016 TomTom base map was linked to the road and appended into the document to finalize it. The reference coordinate system was WGS84 (World Geodetic System 1984). When the document is complete, it was imported into MongoDB under "probe" cluster.

All these steps for preprocessing and import were executed with a python script. For aggregations pandas and numpy, for database connection pymongo libraries were used (see Appendix A for the python code).

3.3.2.2 Pollution data preprocessing

The pollution data includes one metadata file with station and export information, and one value file with hourly values for the complete year. Each export contains only one station and pollutant values. In addition to this, a third file with geometries for all the stations was present.

First, all these station metadata and value files for different stations and pollutants were merged into one metadata and one value file for the ease of data processing. These two files were read together to link correct values to correct stations and to create one document for each station and pollutant combination. This document contains station name, station id, pollutant, data availability percentage and a list of survey values with time and pollution.

Second, the coordinates of the stations in WGS84 datum were appended to the documents from a separate file (i.e. mentioned as third file previously) by linking with the station names. This step was necessary because the pollution data exported from ARPA Lombardia does not contain the geometry. Finally, the documents were imported into MongoDB under "pollution" cluster.

Merging metadata and value files was performed with a batch script, whereas document creation and import were handled with a python script using pymongo library for database connection (see Appendix B for the python code).

3.3.3 Processing and analysis

Once data is implemented, below analyses were tried and tested by running queries on database (see Appendix C for the queries on MongoDB).

3.3.3.1 Individual data statistics for traffic

To have some idea on the data, the averages per hour calculated by querying on MongoDB, using aggregate pipeline. Query can be improved by grouping day (from 6:00 to 18:00) and night (from 18:00 to 6:00), each weekday, working days (i.e. Monday to Friday) and weekends (i.e. Saturday and Sunday), month and season, etc.

3.3.3.2 Individual data statistics for pollution

Similar statistics to traffic data was created also for pollution data, again using MongoDB aggregate functionalities.

3.3.3.3 Spatial querying

For the correlation analysis, the average hourly values for the road edges around stations were needed. First, the road edges around stations selected using "$geonear" function in MongoDB. Second, averages per hour calculated for all the roads around each station. This is a combination of aggregate queries and spatial queries.

3.3.3.4 Correlation analysis between traffic and pollution

According to research done on MongoDB documentation, there is no functionality to calculate correlation in MongoDB. That kind of analysis could be done with python, by calling the results from spatial query with averaged hourly traffic values around stations, and the station values in python to process. As this is not a MongoDB functionality, and not in scope of the aim of this work, that step is skipped.

4. RESULTS

MongoDB appeared to have favorable performance overall, although some results expose areas of improvement.

4.1 Results of Preprocessing of Data

Preprocessing of traffic data was the biggest problem of this study. The preliminary aggregation had to be done in python before importing the data into the database, due to the confidentiality issues of the data and document size limitation in MongoDB. During this preprocess in python, as more slices are preprocessed, runtime for one slice increased exponentially, resulting in an unacceptable processing time. Many aspects reviewed and tested to overcome this issue without any success yet.

Considering that issue is not related to the database and aim of the study is to experiment on the database performance, at the end only a part of the roads inside a polygon in, distriuted among 4 different data files were selected and imported into the database for the sake of the study. After lots of trials, most with most efficient preprocessing method tested, it took 538.96 hours, around 23 days pure processing time in total, only for half of the data.

As an important note to mention, almost all of this time spent for preprocessing. Import into MongoDB took 9.35 min for complete data, all rest of the time spent was for preprocessing.

For pollution data, preprocessing step was quite fast, all preprocessing and import completed in 14.9 seconds.

In Table 4.1 time spend for preprocessing and import per amount of docs can be seen.

Table 4.1: Import times

Data	Count of docs	Total preprocessing time (seconds)	Only import time (seconds)
Traffic data	28613	1940270.26	561.04
Pollution data	43	14.31	0.8

4.2 Results of Processing and Analysis

Statistics for traffic and pollution data were created using aggregate query functions, which are run on one collection, without any join operation. These statistical aggregations were easy to perform and resulted in short times, demonstrating the strength of aggregation queries in MongoDB. While queries on unnested objects were extremely fast, nested objects values were slower.

Spatial querying functions were available in MongoDB, and they performed well for queries on one collection, with geospatial index. Although, geospatial joins across two collections were not possible without any third party tool or script.

Correlation analysis was not achievable with built-in functionalities of MongoDB. Therefore, for this analysis is MongoDB can be only used to retrieve the necessary data. With a test for that kind of action action, also reading and retrieving performance of MongoDB was comprehended as well, which is one of the strongest aspects of it.

Runtimes for queries performed and amount of docs returned can be seen in below Table 4.2.

Table 4.2 : Query runtimes

Query	Count of docs returned	Runtime (Seconds)
export road ids	28613	7.633
count roads	1	7.337
compare total value counts of segments with sum of counts per hour (includes nested values)	28555	185.695
get overall hourly averages (includes nested values)	8784	243.066
get overall hourly averages per pollutant (includes nested values)	68543	0.666
select roads within 50m around point coordinates	8	0.201
select roads within 100m around point coordinates	23	0.462
select roads within 250m around point coordinates	68	0.768
select roads within 500m around point coordinates	88	0.835

4.3 General Results

During the initiation of this study, first database tested was couchdb, but due to limited querying functionalities and insufficient interface, it was decided to change the database to be tested.

Secondly Couchbase is tested. With it, querying and basic functionalities were simpler but imports were excessively slow, therefore database changed once again. It was later understood that the main reason for slow import rate was looping over very big text files during preprocessing, but MongoDB was slightly faster.

After having problems with two NoSQL databases, finally, experiment conducted and completed with MongoDB.

5. CONCLUSIONS

In this work, it was aimed to test the functionalities of a NoSQL database and generally review the performance and capabilities, including geospatial querying options.

As a result of the study, it is observed that aggregate queries and basic spatial queries on single collections are the strength of MongoDB. Whereas, gaps in complex querying and spatial joining functionalities are the weak sides of it. While running very fast for the first level elements in documents, queries on nested elements are giving results much slower. Overall, taking into account that third party softwares can be utilized to perform more complex analyses in MongoDB, it may be asserted that MongoDB is a preferable database for big data management.

The main problem found about MongoDB is the inability of performing complex cross queries on the database. In following works, new ways of querying on MongoDB can be investigated. Also, other NoSQL database systems can be analysed in the same way and supportive tools and engines can be explored in order to develop stronger systems.

Moreover, the dataset created can be used to extract some meaningful results from the data in line with the smart city needs, such as usage of sensor data for decision making and city management, enhancing interpolated pollution models with traffic data, it can be even enriched with sensor data coming from other sources such as meteorological data and open new areas of investment.

REFERENCES

Agarwal, S., & Rajan, K. S. (2017). Analyzing the performance of NoSQL vs. SQL databases for Spatial and Aggregate queries. In Free and Open Source Software for Geospatial (FOSS4G) Conference Proceedings (Vol. 17, No. 1, p. 4).

Baralis, E., Dalla Valle, A., Garza, P., Rossi, C., & Scullino, F. (2017, December). SQL versus NoSQL databases for geospatial applications. In 2017 IEEE International Conference on Big Data (Big Data) (pp. 3388-3397). IEEE.

Chen, M., Mao, S., & Liu, Y. (2014). Big data: A survey. Mobile networks and applications, 19(2), 171-209.

Diaconita, V., Bologa, A. R., & Bologa, R. (2018). Hadoop Oriented Smart Cities Architecture. Sensors, 18(4), 1181.

Hahmann, S., Burghardt, D., & Weber, B. (2011). "80% of All Information is Geospatially Referenced"??? Towards a Research Framework: Using the Semantic Web for (In) Validating this Famous Geo Assertion. In Proceedings of the 14th AGILE Conference on Geographic Information Science.

Lee, J. G., & Kang, M. (2015). Geospatial big data: challenges and opportunities. Big Data Research, 2(2), 74-81.

Li, S., Dragicevic, S., Castro, F. A., Sester, M., Winter, S., Coltekin, A., ... & Cheng, T. (2016). Geospatial big data handling theory and methods: A review and research challenges. ISPRS journal of Photogrammetry and Remote Sensing, 115, 119-133.

Li, Y., & Manoharan, S. (2013, August). A performance comparison of SQL and NoSQL databases. In 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM) (pp. 15-19). IEEE.

Lian, J., Miao, S., McGuire, M., & Tang, Z. (2018, October). SQL or NoSQL? Which Is the Best Choice for Storing Big Spatio-Temporal Climate Data?. In International Conference on Conceptual Modeling (pp. 275-284). Springer, Cham.

Malik, K. R., Sam, Y., Hussain, M., & Abuarqoub, A. (2018). A methodology for real-time data sustainability in smart city: Towards inferencing and analytics for big-data. Sustainable Cities and Society, 39, 548-556.

Zafar, R., Yafi, E., Zuhairi, M. F., & Dao, H. (2016, May). Big data: the NoSQL and RDBMS review. In 2016 International Conference on Information and Communication Technology (ICICTM) (pp. 120-126). IEEE.

Zhang, X., Song, W., & Liu, L. (2014, June). An implementation approach to store GIS spatial data on NoSQL database. In 2014 22nd international conference on geoinformatics (pp. 1-5). IEEE.

Url-1 <*https://www.tomtom.com/lib/doc/licensing/I.CPC.EN.pdf*>, date retrieved 30.04.2019.

Url-2 <*https://www.arpalombardia.it/Pages/Ricerca-Dati-ed-Indicatori.aspx*>, date retrieved 30.04.2019.

Url-3 <*https://www.mongodb.com/what-is-mongodb*>, date retrieved 30.04.2019.

Url-4 <*https://docs.mongodb.com/v3.0/core/sharding-introduction*/>, date retrieved 30.04.2019.

Url-5 <*https://nosqlbooster.com/features#QueryMongoDBwithSQL*>, date retrieved 30.04.2019.

Url-6 <*https://docs.mongodb.com/manual/reference/limits*/>, date retrieved 30.04.2019.

APPENDICES

APPENDIX A: Python codes to process and import traffic data
APPENDIX B: Python codes to process and import station data
APPENDIX C: Queries on MongoDB

APPENDIX A

Python code to create list of beginning and ending line numbers for slices:

```
# -*- coding: utf-8 -*-
"""
Created on Wed May  9 16:18:36 2018

@author: ergin
"""

infiles =
['/media/ezgi/IST_RSO/Milan_Thesis/data/Traffic/00901120e4538368n0l.txt',
        '/media/ezgi/IST_RSO/Milan_Thesis/data/Traffic/00901120e4546560n0l.txt',
        '/media/ezgi/IST_RSO/Milan_Thesis/data/Traffic/00917504e4538368n0l.txt',
        '/media/ezgi/IST_RSO/Milan_Thesis/data/Traffic/00917504e4546560n0l.txt']
#use UTF8 without BOM!
#infiles = ['/media/ezgi/IST_RSO/Milan_Thesis/data/Traffic/samples/sample_.txt']
#use UTF8 without BOM!
out_file = '/media/ezgi/IST_RSO/Milan_Thesis/data/Traffic/segments_20190406.txt'

seg_ids = {}
segments = open(out_file, "a")


### With keys ###
v=len(infiles)
for k in range(v):
   infile=infiles[k]
   with open(infile) as f:
      for ids, line in enumerate(f):
         if line.split(' ')[0] == 'DSEG':
            seg_ids[ids] = line.split(' ')[2]+' '+line.split(' ')[3]
      for i,j in seg_ids.items():
         segments.writelines(str(i) + ' '+ str(j))
   seg_ids = {}

segments.close()
```

Python code to preprocess and import traffic data:

```
# -*- coding: utf-8 -*-
"""
Created on Fri May 25 18:00:20 2018

@author: ergin
"""

import time
```

```
from itertools import islice
import pandas as pd
import numpy as np

start_time = time.time()

##input file path or paths - use UTF8 without BOM!

#infile = '/home/ezgi/Desktop/Milan/sample.txt'
infile = '/home/ezgi/Desktop/Milan/00901120e4538368n0l.txt'
#infile = '/home/ezgi/Desktop/Milan/00901120e4546560n0l.txt'
#infile = '/home/ezgi/Desktop/Milan/00917504e4538368n0l.txt'
#infile = '/home/ezgi/Desktop/Milan/00917504e4546560n0l.txt'
segidfile='/home/ezgi/Desktop/Milan/first_imp_00901120e4538368n0l.txt'
#segidfile = '/home/ezgi/Desktop/Milan/first_imp_00901120e4546560n0l.txt'
#segidfile = '/home/ezgi/Desktop/Milan/first_imp_00917504e4538368n0l.txt'
#segidfile = '/home/ezgi/Desktop/Milan/first_imp_00917504e4546560n0l.txt'
geof = '/home/ezgi/Desktop/Milan/geom.txt'

###mongodb connection
dbname = 'probe'
from pymongo import MongoClient
db=MongoClient('localhost', 27017).milan
collection=db[dbname]


### find segment start index list for all
#seg_ids = {}
#with open(infile) as f:
#    for ids, line in enumerate(f):
#        if line.split(' ')[0] == 'DSEG':
#            seg_ids[ids] = int(line.split(' ')[3])


#or directly give the segids list in dictionary from file
file=open(segidfile, "r").read()
seg_ids=eval(file)
file=""

print("--- %s seconds prep segids ---" % (time.time() - start_time))
print("--- count of segids: ", len(seg_ids))

## slice file for each segment
inp = open(infile, 'r')
geodf = pd.read_csv(geof, sep = '\t', encoding = 'utf-8')
geodf = geodf.drop(columns="Type")
bulk=[]
for i in range(0,len(seg_ids)):
#    start_time_per_seg =time.time()
    ##select the segment
```

```
    inp.seek(0)
    segment_slice = islice(inp,list(seg_ids.keys())[i],
(list(seg_ids.keys())[i]+list(seg_ids.values())[i]+1))
    segment = list(segment_slice)
    ##define header and initiate document
    header = segment[0].split(' ')
    doc = {'seg':str(header[2][1:]),'dir':str(header[2][0]),'tot':int(header[3])}
    ##define contents
    lines = []
    for _, line in enumerate(segment[1:]):
        line_split = line.split(' ')
        ##convert epoche to datetime
        t_orj = int(line_split[1])
        t = time.strftime('%Y-%m-%dT%H:%M:%S', time.localtime(t_orj))
        ##create new line
        new_line = [t, float(line_split[2]), int(line_split[5])]
        lines.append(new_line)
    ##summarize contents
    df=pd.DataFrame(lines, columns=('t','speed', 'cover'))
    lines = []
    df['t'] = pd.to_datetime(df['t'], format='%Y-%m-%dT%H:%M:%S')
    df = df.set_index(pd.DatetimeIndex(df['t']))
    del df['t']
    df_avg = df.resample('H').mean()
    df_count = df.resample('H').count()
    df_a_clean = df_avg.dropna(axis=0, how='all')
    df_c_clean = df_count.replace(0,np.nan).dropna(axis=0, how='all')
    df_final = pd.concat([df_a_clean.speed, df_c_clean.cover],
axis=1).rename(columns={'speed': 'avg_speed', 'cover': 'count'})
    df = []
    df_avg = []
    df_count = []
    df_a_clean = []
    df_c_clean = []

    ##append summary into document as list
    segment_tags =[]
    for j in df_final.index:
        d = {"tm": str(j),
"spd":float(df_final['avg_speed'][j]),"cnt":int(df_final['count'][j])}
        segment_tags.append(d)
    doc.update({"values":segment_tags})
    segment_tags =[]
    df_final = []

    geofltr = geodf['segid'] == header[2][1:]
    g = {'geometry':{'type':"LineString",
'coordinates':eval(list(geodf[geofltr]['coordinates'])[0])}}
    doc.update(g)
    bulk.append(doc)
```

```
    doc=[]
    if len(bulk)>1000:

        print("--- %s seconds bulk prep complete---" % (time.time() - start_time))
        print("--- count of segments: ", len(bulk))

        ##bulk insert into mongodb
        collection.insert_many(bulk)

        print("--- %s seconds bulk insert complete---" % (time.time() - start_time))
        bulk=[]
    else:
        continue


print("--- %s seconds last bulk prep complete---" % (time.time() - start_time))
print("--- count of segments: ", len(bulk))

##bulk insert into mongodb
collection.insert_many(bulk)

print("--- %s seconds bulk insert complete---" % (time.time() - start_time))
```

APPENDIX B

Python code to preprocess and import pollution data:

```
# -*- coding: utf-8 -*-
"""
Created on Fri May 25 18:00:20 2018

@author: ergin
"""

import time
start_time = time.time()

##input file path or paths - use UTF8 without BOM!
values =
'/media/ezgi/IST_RSO/Milan_Thesis/data/Pollution/Station_values/import_values.txt
'
legenda =
'/media/ezgi/IST_RSO/Milan_Thesis/data/Pollution/Station_values/import_legenda.t
xt'
geof =
'/media/ezgi/IST_RSO/Milan_Thesis/data/Pollution/Station_values/Stazioni_scope_g
eom.geojson'

###mongodb connection
dbname = 'pollution'
from pymongo import MongoClient
db=MongoClient('localhost', 27017).milan
collection=db[dbname]

geom=eval(open(geof).read())['features']
del geof

heads=[]
with open(legenda) as l:
    for idl, rowl in enumerate(l):
        header=rowl.split('\t')
        head={'st_id':str(header[0]),
              'st_name':str(header[1].replace("Ã\xa0","a").replace(" - "," ").replace(" P.zza
","  ").replace(" viale "," ").replace(" via "," ").replace("   "," ")),
              'sensor_id':str(header[2]),
              'pollutant':str(header[4]),
              'formula':str(header[5]),
              'availability':str(header[6]),
              'unit':str(header[7]).replace('Â','').replace('\n','')}
        for i in geom:
            st_ng = str(i['properties']['Stazione']).replace("Ã\xa0","a")
            if st_ng==str(head['st_name']):
                head.update({'geometry':i['geometry']})
```

```
        heads.append(head)

print("--- %s seconds prep headers ---" % (time.time() - start_time))

for i in range(1,len(heads)):
    start_time2 = time.time()
    doc=heads[i]
    sensorid=doc['sensor_id']
    st_tags=[]
    with open(values) as v:
        for idv, rowv in enumerate(v):
            line=rowv.split('\t')
            if str(sensorid)==str(line[0]):
                d={"tm":str(line[1]),"poll":float(line[2])}
                st_tags.append(d)
    doc.update({"values":st_tags})
    st_tags=[]
    ##insert into mongodb
    collection.insert_one(doc)
    print("--- %s seconds --- per doc" % (time.time() - start_time2))



print("--- %s seconds import end ---" % (time.time() - start_time))
```

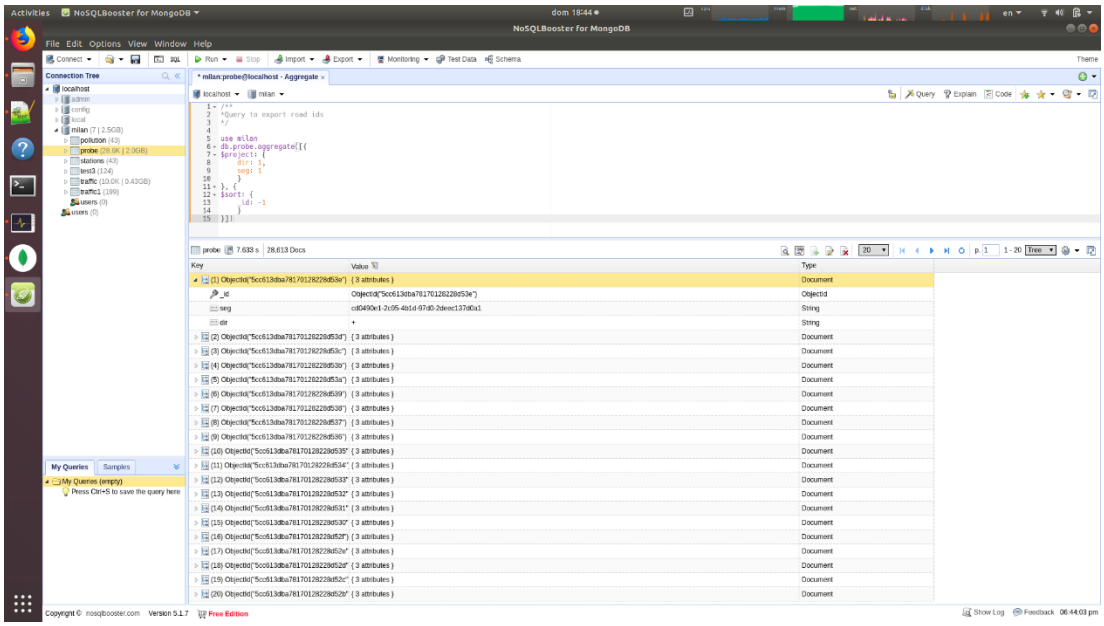
APPENDIX C

```
/**
*Query to export road ids
*/

use milan
db.probe.aggregate([{
$project: {
        dir: 1,
        seg: 1
        }
}, {
$sort: {
        _id: -1
        }
}])
```

Runtime: 7.633

==========

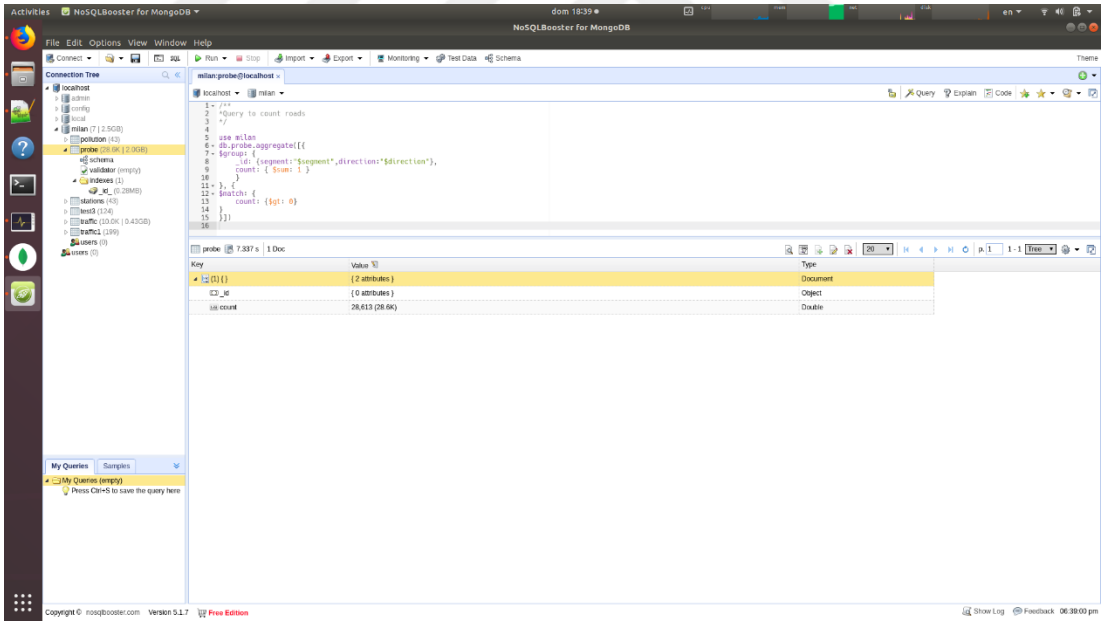


```
/**
*Query to count roads
*/

use milan
db.probe.aggregate([{
$group: {
        _id: {segment:"$segment",direction:"$direction"},
        count: { $sum: 1 }
        }
}, {
$match: {
        count: {$gt: 0}
        }
}])
```

Runtime: 7.337

==========



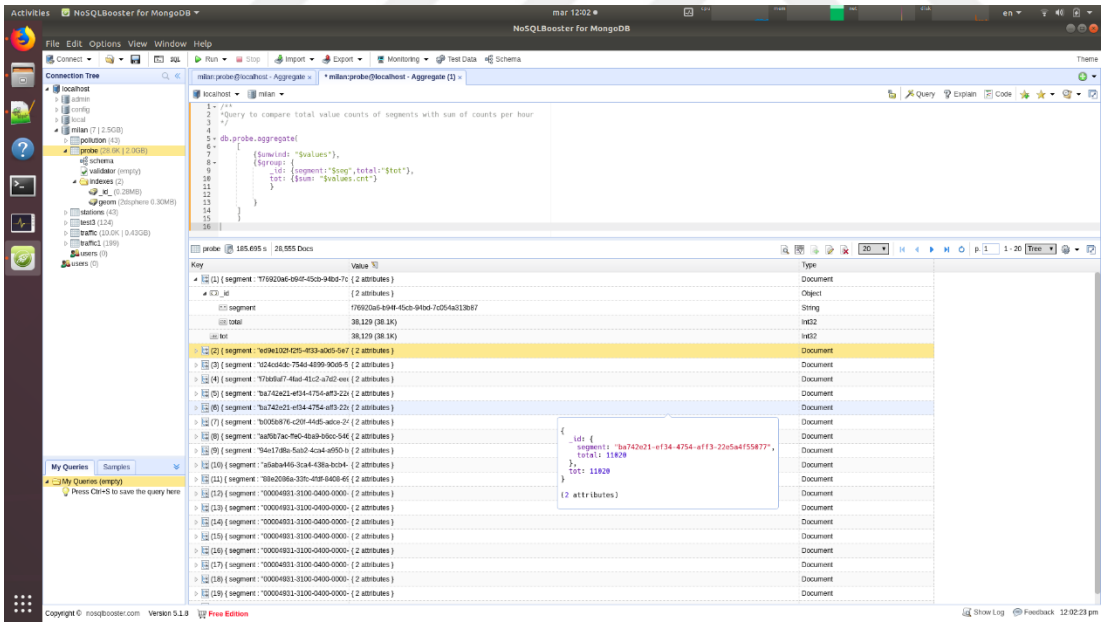
```
/**
*Query to compare total value counts of segments with sum of counts per hour
*/

db.probe.aggregate(
	[
	{$unwind: "$values"},
	{$group: {
	_id: {segment:"$seg",total:"$tot"},
	tot: {$sum: "$values.cnt"}
	}

	}
	]
	)
```

Runtime: 185.695

==========



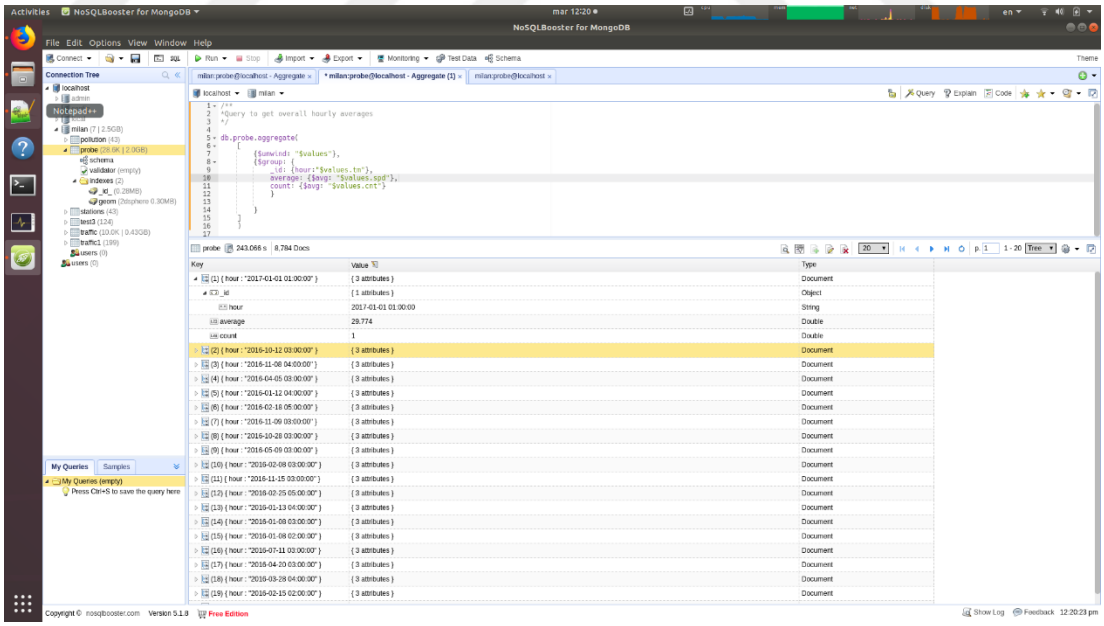
```
/**
*Query to get overall hourly averages
*/

db.probe.aggregate(
        [
        {$unwind: "$values"},
        {$group: {
        _id: {hour:"$values.tm"},
        average: {$avg: "$values.spd"},
        count: {$avg: "$values.cnt"}
        }

        }
        ]
        )
```

Runtime: 243.066

==========



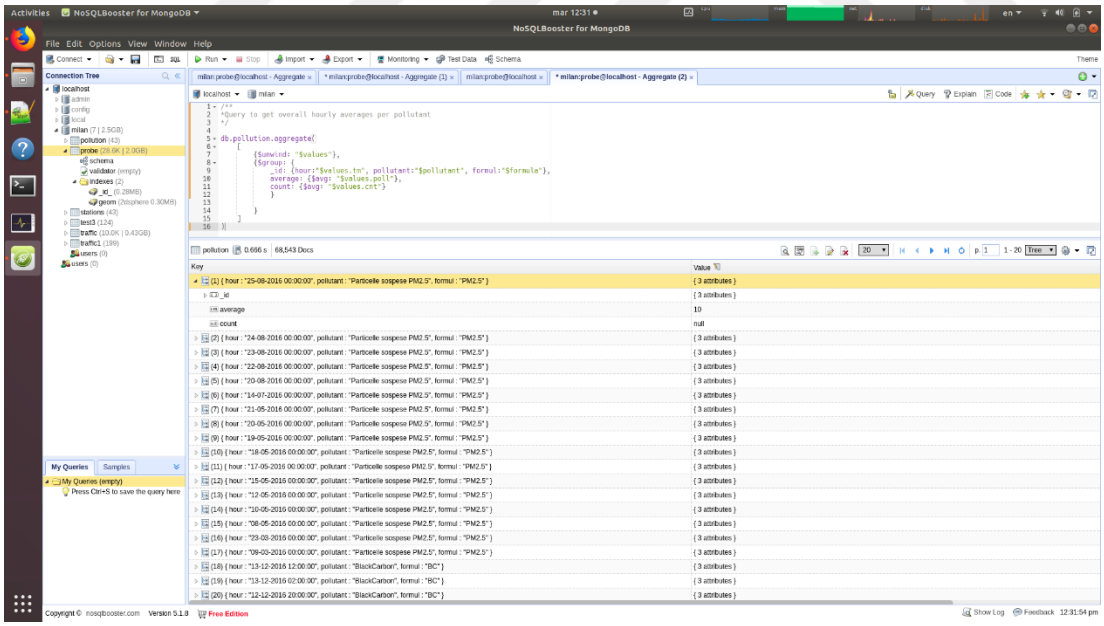
```
/**
*Query to get overall hourly averages per pollutant
*/

db.pollution.aggregate(
        [
        {$unwind: "$values"},
        {$group: {
        _id: {hour:"$values.tm", pollutant:"$pollutant", formul:"$formula"},
        average: {$avg: "$values.poll"},
        count: {$avg: "$values.cnt"}
        }

        }
        ]
)
```

Runtime: 0.666

==========

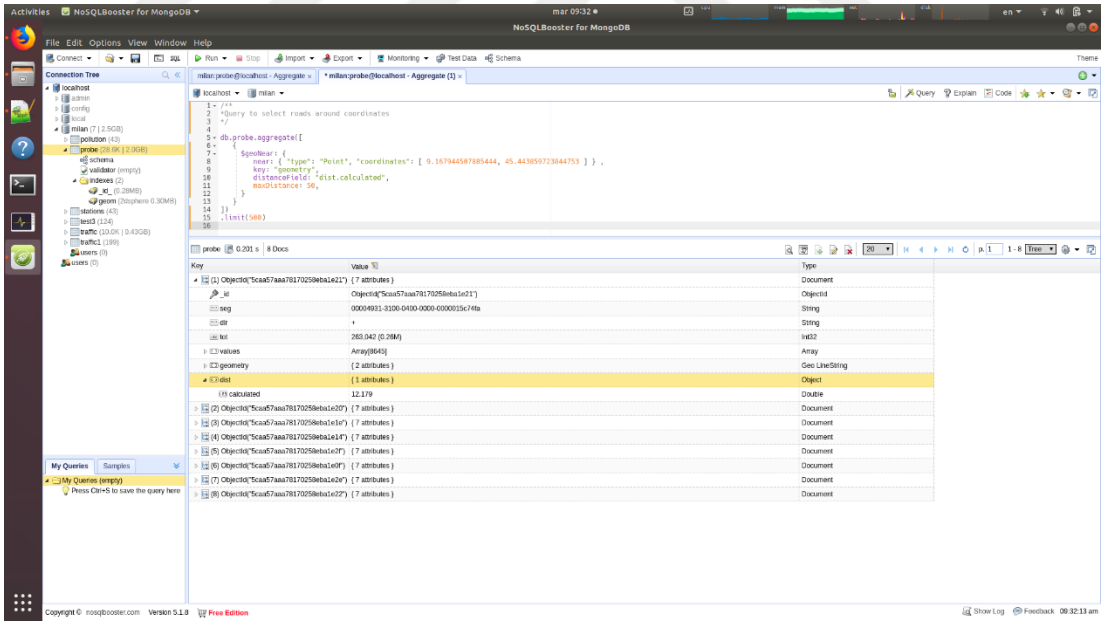


```
/**
*Query to select roads around coordinates
*/

db.probe.aggregate([
  {
        $geoNear: {
        near: { "type": "Point", "coordinates": [ 9.167944507885444,
45.443859723844753 ] } ,
        key: "geometry",
        distanceField: "dist.calculated",
        maxDistance: 50,
        }
  }
])
.limit(500)
```

Runtime: 0.201

==========

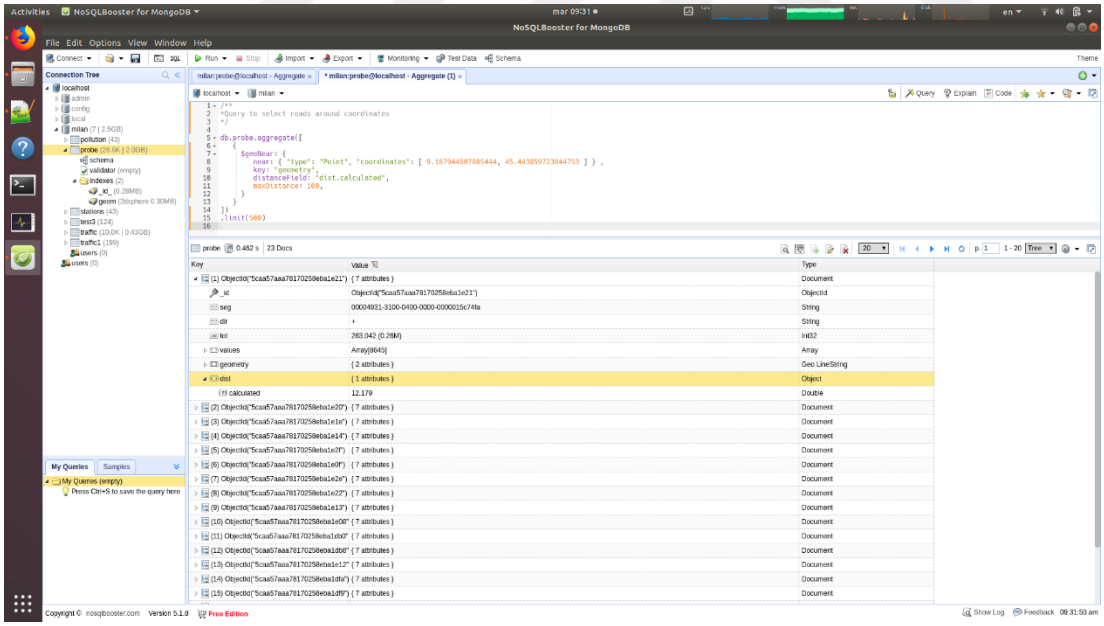


```
/**
*Query to select roads around coordinates
*/

db.probe.aggregate([
  {
        $geoNear: {
        near: { "type": "Point", "coordinates": [ 9.167944507885444,
45.443859723844753 ] } ,
        key: "geometry",
        distanceField: "dist.calculated",
        maxDistance: 100,
        }
  }
])
.limit(500)
```

Runtime: 0.462

==========



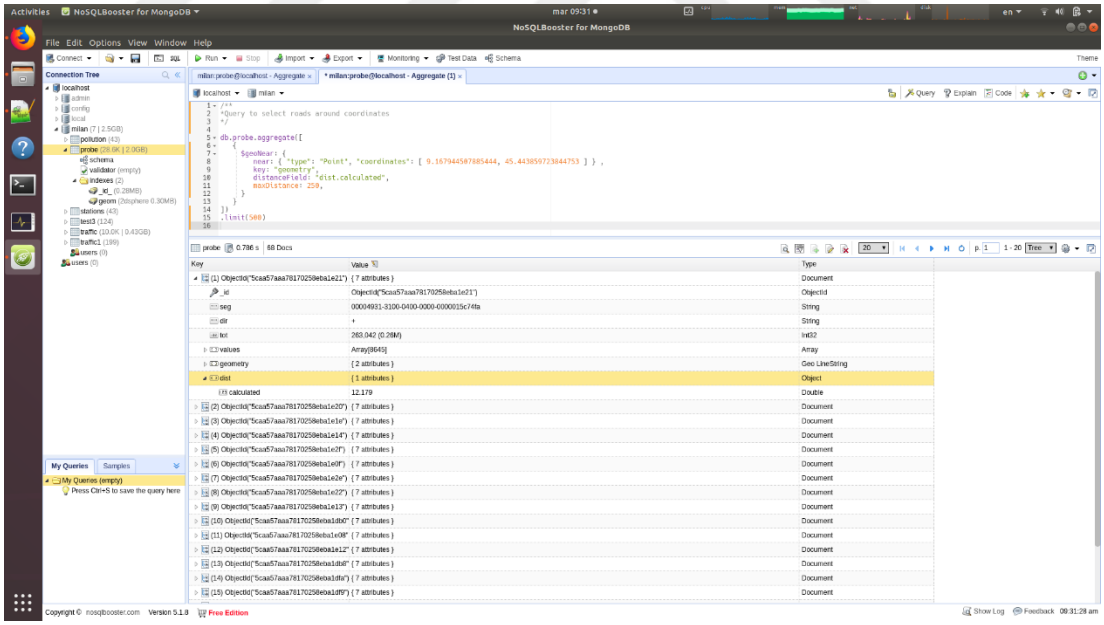
```
/**
*Query to select roads around coordinates
*/

db.probe.aggregate([
  {
        $geoNear: {
        near: { "type": "Point", "coordinates": [ 9.167944507885444,
45.443859723844753 ] } ,
        key: "geometry",
        distanceField: "dist.calculated",
        maxDistance: 250,
        }
  }
])
.limit(500)

-----------------------------------------------------------------------------------

Runtime: 0.768

==================================================
```

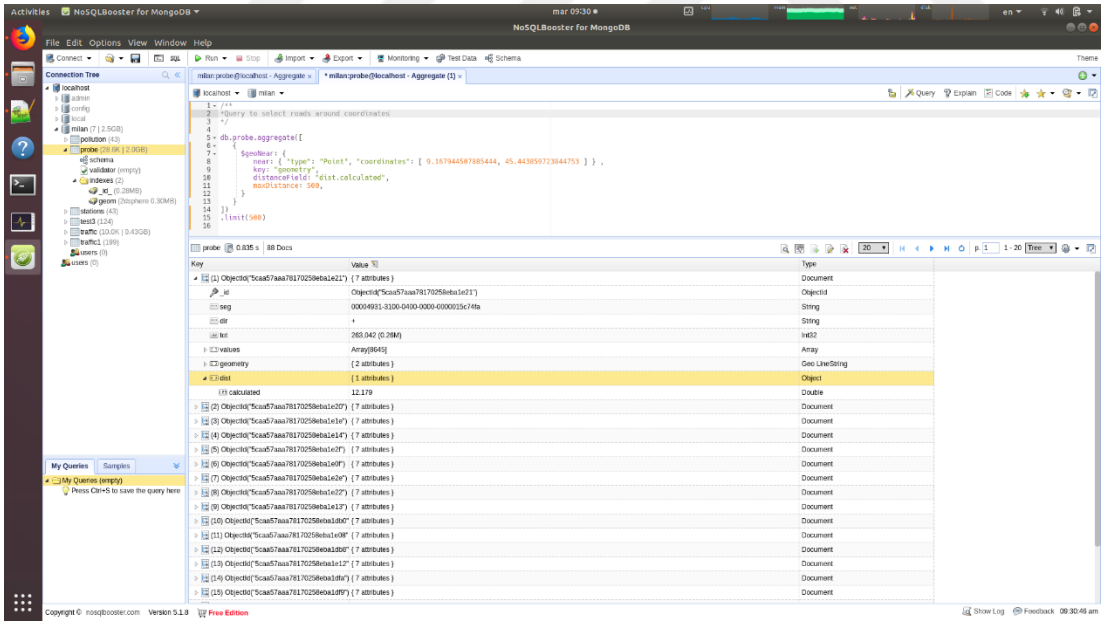


```
/**
*Query to select roads around coordinates
*/

db.probe.aggregate([
  {
        $geoNear: {
        near: { "type": "Point", "coordinates": [ 9.167944507885444,
45.443859723844753 ] } ,
        key: "geometry",
        distanceField: "dist.calculated",
        maxDistance: 500,
        }
  }
])
.limit(500)
```

Runtime: 0.835

==========



CURRICULUM VITAE

Name Surname: Ezgi Ergin

Place and Date of Birth: İstanbul, 25.01.1989

Address: Gent, Belgium

E-Mail: ezgiergin89@gmail.com

B.Sc.: 2010, İstanbul Technical University, Civil Engineering Faculty, Department of Geomatics Engineering

Professional Experience and Rewards:

- 2012- …. TomTom International BV., Quality, Training and Methods Coordinator
- 2011-2012 Sistem AŞ, GIS products Sales & Support

List of Publications and Patents:

- Ergin, E., Dogru, A. O. (2019): "An Investigation on Geospatial Functionalities of MongoDB", Submitted to International Symposium on Applied Geoinformatics, Istanbul, Turkey.
- "Free & Open Source GIS Technology", ITU Geomatics Engineering, BSc. Thesis, Prof. Dr. Rahmi Nurhan Çelik, Dr. Caner Güney, August 2010.