SQL versus NoSQL Databases for Geospatial Applications

SQL versus NoSQL Databases for Geospatial Applications

Elena Baralis, Andrea Dalla Valle, Paolo Garza
Dipartimento di Automatica e Informatica
Politecnico di Torino
Torino, Italy
elena.baralis@polito.it
andrea.dallavalle@studenti.polito.it
paolo.garza@polito.it

Claudio Rossi and Francesco Scullino
Microsoft Innovation Center
Istituto Superiore Mario Boella (ISMB)
Torino, Italy
rossi@ismb.it
scullino@ismb.it

***Abstract*—In the last years, we are witnessing an increasing availability of geolocated data, ranging from satellite images to user generated content (e.g., tweets). This big amount of data is exploited by several cloud-based applications to deliver effective and customized services to end users. In order to provide a good user experience, a low-latency response time is needed, both when data are retrieved and provided. To achieve this goal, current geospatial applications need to exploit efficient and scalable geospatial databases, the choice of which has a high impact on the overall performance of the deployed applications. In this paper, we compare, from a qualitative point of view, four state-of-the-art SQL and NoSQL databases with geospatial features, and then we analyze the performances of two of them, selecting the ones based on the Database-as-a-service (DBaaS) model: Azure SQL Database and Azure DocumentDB (i.e., an SQL database versus a NoSQL one). The empirical evaluation shows pros and cons of both solutions and it is performed on a real use case related to an emergency management application.**

***Keywords*-Big Geospatial data; Geospatial databases; Database-as-a-service (DBaaS)**

I. Introduction

The availability of big geospatial and geolocated data significantly increased in the last years [1], [2] and many applications have been proposed to exploit geospatial data to provide innovative services tailored to users needs in different domains (e.g., smart cities, IoT, emergency management, archeology [3], [4], [5], [6], [7], [8]). In order to provide a good user experience, efficient geospatial databases are needed. For this reason, we evaluate state-of-the-art geospatial databases in order to understand which are the pros and cons of the current solutions. We consider both SQL and NoSQL databases [9] to understand which category of databases is more efficient for managing large geospatial data. Specifically, we initially compare four state-of-the-art geospatial databases by considering only their features. Then, we perform a large set of experiments to evaluate the efficiency of the selected geospatial databases by means of a real application, which has been designed for supporting emergency management services related to flood, fire, and extreme weather events. Since there is an increasing interest in cloud-based solutions, the application is developed using the Software-as-a-Service (SaaS) approach and also the geospatial database is deployed in the cloud using such approach. This configuration allows us to evaluate the efficiency of geospatial databases when they are deployed in a cloud-based system with the SaaS service model, which is becoming increasingly adopted, against operational costs. The basic functionalities of the implemented application are similar to those of several other developed in different contexts. Specifically, it allows the end users to retrieve or send geolocated data based on their location, which is one of the standard behaviors of a general location based service. The performed experiments highlight pros and cons of the analyzed geospatial databases.

The main contributions of this paper are (i) a qualitative comparison of geospatial databases, (ii) a performance comparison of two them in a real use case, i.e., a Cloud-based SaaS deployment, and (iii) a summary of pros and cons of the evaluated SQL and NoSQL solutions.

The paper is organized as follows. Section II describes the related work, while Section III analyzes the main features of the four state-of-the-art geospatial databases and compare them from a qualitative point of view. Section IV reports a case study, an emergency management geospatial application that is used in Section V to evaluate the performances of SQL and NoSQL geospatial databases from different point of views. Finally, Section VI draws conclusions and describes future work.

II. Related Work

Current databases are usually classified in two macro-categories: relational databases (e.g., [10], [11], [12]) and NoSQL databases (e.g., [13], [14], [15], [16]). Relational databases are usually the most appropriate and efficient choice when data are structured (i.e., when data are characterized by a known a-priori and fixed set of attributes known at design time). Differently, NoSQL databases, such as key-value-based and document-oriented databases, are usually more appropriate when the input data collection contains unstructured or semi-structured data, or when the structure

(attributes) of the data evolves overtime and it is (partially) unknown at design time. The majority of the traditional databases are mainly focused on non-spatial data. However, since such type of data play an important role in many application domains, several spatial databases, relational and not, have been proposed to manage and query geospatial data [10], [11], [13], [14]. Spatial databases, or in general databases with (geo)spatial features, are usually an extension of traditional databases in which ad-hoc (geo)spatial data types, query operators and indexes are integrated. The most common use of geospatial databases consists in executing queries that select all the objects contained in a geographical area or the k-Nearest Neighbours of an input object. Some of the most famous and commonly used state-of-the-art relational databases with geospatial features are SQL Server 2016 [17] and its cloud version, which is also commercially named Azure SQL Database [10], and PostGIS [11], while two well established NoSQL databases with geospatial features are DocumentDB [13] and MongoDB [14].

The advantages and disadvantages of relational vs non-relational databases have been throughly analyzed in many papers [18], [9]. However, the reported analyses are not focused on the geospatial features of the selected databases. In this paper, we perform a qualitative and quantitative analyses of the geospatial characteristics, comparing SQL vs no-SQL databases.

In recent years, we witnessed an increasing interest in cloud-based applications deployed using the Database-as-a-service (DBaaS) model [19]. When the database is provided as a service, developers/application owners do not have to install and maintain the databases used by their applications. The provider offers a pre-defined set of service levels, each of which is generally associated to a monthly costs that includes the hosting, the installation and the maintenance operations. In this way, the developers/application owners can focus on the design of the database without worry about software and hardware failures and updates. For instance, Microsoft offers both SQL [10] and NoSQL [13] databases as services. However, several other vendors provide similar services (e.g., Amazon Web Services and Oracle). Moreover, the Database-as-a-service model allows increasing or decreasing the performances of the database on demand, depending on the current work load of the system. The performance analysis of DBaaS databases in combination with location based services deployed using HTTP web services is a practical yet interesting topic that has not been addressed in previous work, and that we explore in this paper.

III. Qualitative Comparison of Databases with Geospatial Features

In this section, we analyze the main characteristics of four state-of-the-art databases with geospatial features. Specifically, we selected two SQL and two NoSQL databases. For each of them we highlight pros and cons, from a qualitative point of view, by taking into consideration also the application scenario in which they are generally used.

A. Relational databases

Relational databases (e.g., [17], [10], [11], [12]) are well-established systems used for storing and querying large structured data sets. They are based on the relational model, i.e., a set of tables where each table is used to represent a set of objects with similar characteristics. They have been successfully used in both Online transaction processing (OLTP) and Online Analytical Processing (OLAP) applications. Many solutions have been proposed for increasing the efficiency of relational databases, some of which are also characterized by geospatial features (e.g., [17], [10], [11]). Specifically, indexes and partitioning are commonly used to boost their performances. In the following, we describe two of the most commonly used relational databases with geospatial features: Azure SQL Database [10] and PostGIS [11].

1) Azure SQL Database: Azure SQL Database [10] is a relational Database-as-a-Service (DBaaS). It has the same functionalities of the centralized Microsoft SQL Server 2016 [17] but it is provided as a service in the cloud. It can be easily and dynamically adapted to the load of applications by using a scale-up approach. Through a simple graphical interface, the database administrator can increase the power of the server that is used to run the database, and the Azure system transparently increases the performances of the database while avoiding service interruptions and data losses. Both Azure SQL Database and Microsoft SQL Server 2016 allow defining and quering spatial objects. They support two main categories of data types: (i) Geometry, which represents data on a Euclidean coordinate system by using flat XY coordinate pairs and (ii) Geography, which represents data on an earth-like spherical coordinate system in longitude-latitude shape. Moreover, Azure SQL Database can manage the standard spatial objects (e.g., Point, MultiPoint, Polygon, MultiPolygon) and standard spatial operations available for geography objects (e.g., Intersection, Distance, Difference, Within). Specifically Azure SQL Database implements the Simple Features for SQL specification from the Open Geospatial Consortium (OGC) [20], [21]. Azure SQL Database and Microsoft SQL Server 2016 support also spatial indexes, which are implemented using the B-Tree (i.e., Binary Tree) data structure. Azure SQL Database and Microsoft SQL Server 2016 are compatible with state-of-the-art geographic information systems (GIS) such as ArcGIS [22] as well as with map server softwares such as GeoServer [23]. Both tools are commonly used to analyze geospatial data and visualize maps by exploiting geospatial queries and data types.

2) PostGIS: PostGIS [11] is a spatial database extender of the open source relational PostgreSQL database [12]. It

enables the definition of geospatial objects and operations by extending the basic data types and querying operators of PostgreSQL. PostGIS follows the Simple Features for SQL specification from the Open Geospatial Consortium (OGC) [20], [21]. PostGIS is the most efficient open source solution for managing geospatial data. Similarly to Azure SQL Database, it supports all the standard geometry data types (e.g., Point, MultiPoint, Polygon, MultiPolygon) and all the standard geospatial operators (e.g., Distance, Within, Intersects, Closest). It also supports three types of spatial indexes: B-trees (binary trees), R-trees (sub-rectangles trees) and GiST (Generalized Search Trees) to speed up the execution of spatial queries. Also PostGIS is compatible with state-of-the-art geographic information systems (GIS) such as ArcGIS [22] as well as with map server softwares such as GeoServer [23].

B. NoSQL databases

NoSQL databases (e.g., [13], [14], [15], [16]) are commonly used to manage unstructured data, such as documents. They are designed for scaling horizontally because they have been proposed to manage big data sets by means of commodity servers. However, in order to obtain horizontal scalability, they do not provide the standard ACID properties that are usually provided by relational databases. Depending on the application requirements, this could be a weakness of NoSQL databases. Only few NoSQL databases support geospatial data. MongoDB and Azure DocumentDB are two NOSQL document databases that are commonly used in many big data applications and provides geospatial functionalities.

1) MongoDB: MongoDB [14] is an open source NoSQL document-oriented database, based on JSON-like documents. To perform queries and for storage purposes on geospatial data, MongoDB needs an initial definition of the surface type used for running operations on its data. It supports two surfaces: (i) Spherical, which involves calculation based on an Earth-like sphere and (ii) Flat, which considers a Euclidean plane with 2d coordinates, stored as legacy coordinate pairs (i.e., pairs of longitude, latitude values).

MongoDB supports a set of standard GeoJSON data types (Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, and GeometryCollection) and implements a subset of basic spatial operations (inclusion, intersection, and proximity). Hence, the types of possible queries are limited with respect to those provided by the relational state of the art spatial databases, such as Microsoft Azure SQL Database and PostGIS. Moreover, MongoDB supports only basics 2-dimensional indexes.

MongoDB supports also horizontal scalability (i.e., the scalability achieved by adding new commodity servers in a cluster environment when the size of the data increases) and distributed execution of queries by exploiting the sharding technique. The basic idea exploited by the sharding technique consists in partitioning the input data collection in chunks and store each chunk on a different server. When a query is executed, each server executes the query on its chunk of data, parallelizing its execution. The partitioning of the data is based on the value of the selected sharding attribute. Hence, the choice of the sharding attribute is crucial in order to achieve a balance distribution of the data in the servers. It is important to highlight that MongoDB supports also the use of geospatial attributes as sharding attribute. The selection of the most appropriate sharding attribute is based on the predicates of the (expected) frequent queries. MongoDB provides no automatic support for changing the sharding attribute after sharding a collection. Hence, a good estimation of the expected work load (i.e., types of expected queries) at design time is important. The connection with the GeoServer is available through an external plug-in included in the GeoTools suite but it is not officially supported.

2) Azure DocumentDB: DocumentDB [13] is a NoSQL document-oriented database designed by Microsoft. Similarly to Azure SQL Database, also DocumentDB is available as a service. It can manage large data collections by distributing data on a set of servers by means of the sharding approach. DocumentDB implements the same spatial operations and data types supported by MongoDB and it is compatible with the protocol used by MongoDB. Hence, the applications written for MongoDB can use DocumentDB as data store by using the existing drivers for MongoDB and changing the connection string. Equivalently to MongoDB, there is the unsupported plug-in included in the GeoTools to connect DocumentDB to the GeoServer.

C. Comparison of Databases with Geospatial features

In Sections III-A1-III-B2, we described the main features of four state-of-the-art databases with geospatial features. In this section, we report a summary of the functionalities of the considered databases and draw conclusions about their pros and cons. The summary of the qualitative comparison of the selected databases with geospatial features is reported in Table I. The comparison is based on the following key features:

- Types of supported geometry objects
  - Types of objects that can be represented (e.g., points, lines, polygons).
- Implemented geometry functions
  - Types of analysis and queries that can be executed by means of built-in functions.
- Spatial index support and types of supported indexes
  - The availability of indexes allows enhancing the performance of the queries.
- Compatibility with GeoServer
  - GeoServer is a commonly used open source map server software used to share, process and edit geospatial data. The compatibility with it, or a

Table I
QUALITATIVE COMPARISON OF THE FUNCTIONALITIES OF FOUR STATE-OF-THE-ART GEOSPATIAL DATABASES.

Database	**Supported Geometry objects**	**Main supported geometry functions**	**Supported Spatial indexes**	**Compatibility with GeoServer**	**DaaS**	**Horizontal scalability**
PostGIS	Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeometryCollection	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, Distance/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, Distance/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)

similar system, is indispensable to deliver the content to the end users on maps, especially when bandwidth constrained mobile devices are used.

- Database as a Service (DaaS)
  - The availability of the database in the DaaS version allows hiding the complexity of the database administration, demanding such activities to the service provider.
- Horizontal scalability
  - A database characterized by a horizontal scalability can easily scale with respect to the number of requests by including more (commodity) servers. This feature is extremely useful when managing big data.

The main difference that is highlighted by the information reported in Table I is that relational databases (PostGIS and Azure SQL) implement more geospatial functionalities than NoSQL ones (MongoDB and DocumentDB). Moreover, they are also more tightly integrated and supported by the GeoServer software. In fact, the plug-in that is used to connect GeoServer with MongoDB and DocumentDB is in the unsupported branch of the current version of GeoTools [24]. The interoperability with the GeoServer, or with a similar software, is indispensable for geospatial applications that need to visualized maps and geolocated objects, especially when mobile devices are involved. Hence, in terms of functionalities, spatial relational databases are preferable (they are a more mature technology) because they allow performing complex geospatial queries and are well-integrated with geographic information systems (GIS) as well as with map server software such as GeoServer.

In the Section V, we will perform a set of experiment to evaluate also the efficiency of the selected databases in a real application scenario.

IV. CASE STUDY: AN EMERGENCY MANAGEMENT APPLICATION

To evaluate the efficiency of the selected spatial databases, we test two web services that are used by a real emergency management application. Specifically, we implement two services that can be used by first responders (e.g., civil protection agents) and common citizens in case of natural hazards to send geolocated reports from a mobile device in order to improve the real-time understanding of the situation in the field, thus enhancing the assessment of the situational picture and hence improving the decision making

process. The application allows both authorities and citizens to generate a report at a specific location (point) and to retrieve all reports available on a given geographical region. Each report is a characterized by a description (a text field) and a geolocated picture.

The described application is based on two web services:

- GET web service: this service is used to retrieve all the reports located in a specific area, which is characterized by a rectangular bounding box.
- POST web service: this service is used to submit a geolocalized report, with the associated text and picture.

The considered use case and the implemented services are similar to those analysed in previous studies [25], and they are common in many geospatial applications, which usually allow retrieving data on a given geographical area and submitting data with the associated position. Hence, the results reported in Section V, which are based on the emergency management application, can be used as reference also to chose the database for other geospatial applications.

In the following section, we analyze the performance of the two implemented web services by using two different databases.

V. Experimental results

We performed an experimental campaign to evaluate the efficiency of SQL with respect to NoSQL geospatial databases in the real application context described in Section IV. The application has been implemented by means of a set of web services deployed by using the Platform-as-a-Service (PaaS) paradigm. Specifically, Microsoft Azure Cloud Service has been used to deploy the application. We performed the experiments by considering an SQL database and a NoSQL one. Specifically, we selected Azure SQL Database and Azure DocumentDB, respectively. We decided to select those databases because they are tightly integrated with the Microsoft Azure Cloud infrastructure and are provided as services (Database-as-a-service (DBaaS)). In this way we avoid the introduction of biases in the evaluation due to different service provider performances. Since they are based on the DBaaS model, we can easily change the power of the hardware that is used to run the database. This allows us to clearly understand also the database resource requirements for the implemented geospatial application.

Since the price of the (DBaaS) databases is related to the underlying resources, the analysis of the database "performance level" in function of its cost is an interesting practical information. To perform this analysis, for each database we consider three different configurations, hence different performance levels. The selected configurations, their main characteristics, and the related monthly prices[1] are reported in Table II. A set of predefined configurations are provided

[1]The reported prices could be different from the current ones.

Table II
Qualitative comparison of the functionalities of four state-of-the-art geospatial databases.

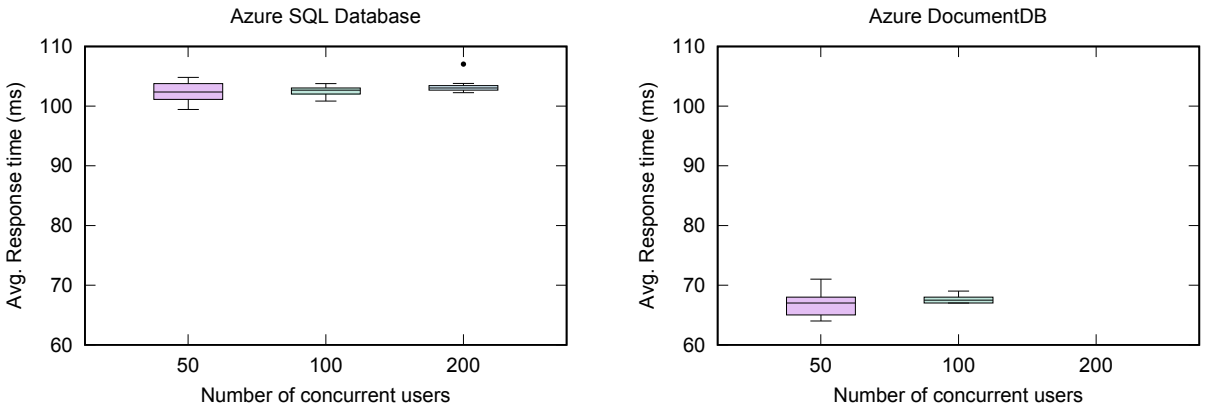
Azure SQL Database Service Tier / Performance level	**Azure DocumentDB RUs / Performance level**	**Monthly price (euro)**
Tier P2	13500 RUs	∼800 euro
Tier P4	29200 RUs	∼1500 euro
Tier P6	60400 RUs	∼3100 euro

for Azure SQL Database, whereas Azure DocumentDB is more flexible and allows setting the maximum performance of the selected database by specifying the maximum number of requested Request Units (RUs). The higher the RU value, the better the performance of the database. Specifically, there is a predefined set of Azure SQL Database configurations, each one characterized by a specific maximum value of Database Transaction Units (DTUs), where the number of DTUs represents the maximum performance of the used database. Differently, Azure DocumentDB is characterized by a different performance measure, called Request Units (RUs), and the end user can specify the desired "performance level" of Azure DocumentDB by setting the number of Request Units (RUs). DTUs and RUs are not directly comparable. Hence, in order to perform a fair comparison, we decided to select a set of configurations of the two databases characterized by the same monthly price. Specifically, for each predefined configuration of Azure SQL Database, we selected an Azure DocumentDB configuration characterized by approximately the same monthly price by setting the number of RUs value. The monthly price is probably the most important factor when selecting a cloud-based service, including the selection of a DBaaS database, because the users of cloud-based services are interested in understanding what is the minimum price that allows satisfying their application requirements (in terms of execution time and scalability). From a different points of view, given two DBaaS databases characterized by the same monthly price, the service users are interested in selecting the most efficient and scalable one.

To perform the evaluation campaign we used a set of synthetic data sets, which are publicly available at http://dbdmg.polito.it/GeoSpatialData/. The cardinality (number of records) of the considered data sets ranges from 250,000 records to 1,000,000 records. We generated two versions of each data set, characterized by the same content but a different data format. Specifically, we generated a relational and a GeoJSON version of each data set, which are used to test Azure SQL Database and Azure DocumentDB, respectively.

A. Performance evaluation

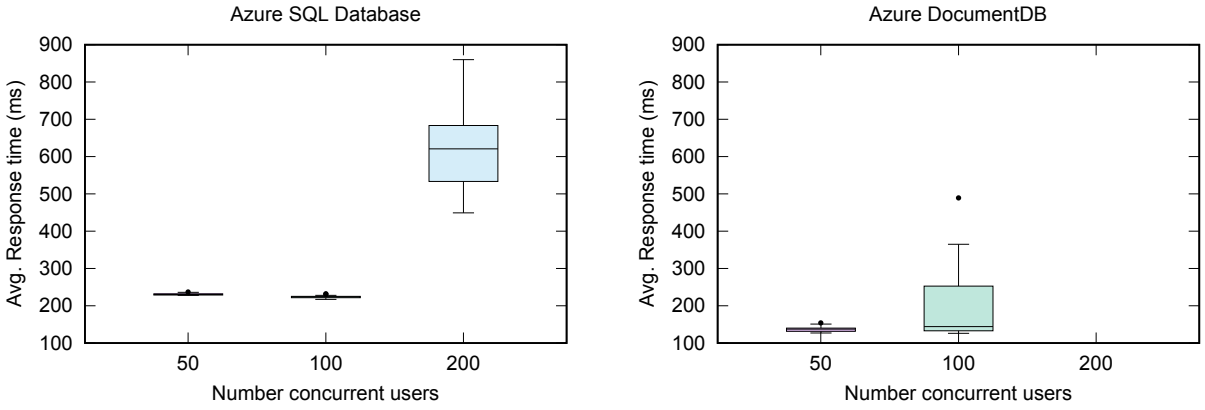
We evaluated the performances of the two selected databases, in terms of average response time per request, with respect to three main factors: (i) the number of con-



(a) Azure SQL Database - Tier P6.

(b) Azure DocumentDB - 60400 RUs.

Figure 1. GET operation. Average response time with respect to the number of concurrent users. Number of records = 1,000,000.



(a) Azure SQL Database - Tier P6.

(b) Azure DocumentDB - 60400 RUs.

Figure 2. POST operation. Average response time with respect to the number of concurrent users. Number of records = 1,000,000.

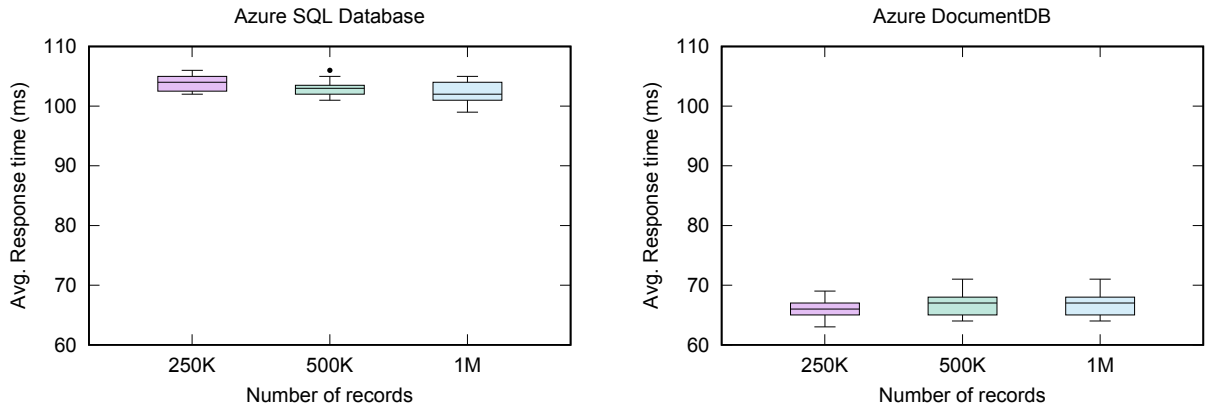
current users submitting requests, (ii) the number of records stored in the database, and (iii) the database performance level.

We performed the tests by using a specific functionality of Visual Studio Enterprise. Specifically, we used the Visual Studio Online stress tool that allows specifying (i) which web services must be invoked, (ii) by how many concurrent users, and (iii) the users' behavior. This tool simulates the invocations of services to assess the performances web service-based applications. All the performed tests are characterized by a warm-up time of 1 minute, during which no measurements are made, and then 5 minutes of effective test. This setting allows excluding sampling in the transient period and measure the performances at regime. For all experiments, we set iteration time to 2 seconds, which defines the awaiting time between a request and another one from the same user. This setting enable the developer to simulate the gap of time when a user starts two different requests (or the same, if stuck).

We tested a "GET web service" that simulates the request of a set of reports located in a user specified area (i.e., a select request that returns a subset of the database records) and a "POST web service" that simulates the upload of a report by the end user (i.e., an insert request that insert a new record in the database). Based on the size of the bounding box of the area of interested that we selected, the select operation executed by the invocation of the GET web service returns 20 reports. This value is comparable to that attended when real data are used.

The results of the performed experiments are reported in the following subsections.

1) Impact of the number of concurrent users: The first set of experiments aimed at analyzing the impact of the number of concurrent users using the developed services and hence the underlying database. We considered a number of users ranging from 50 to 200 users. We performed two separate sets of load tests: one for the GET web service (i.e., we simulate a user who requests all the reports geolocated



(a) Azure SQL Database - Tier P6.

(b) Azure DocumentDB - 60400 RUs.

Figure 3. GET operation. Average response time with respect to the number of records. Number of concurrent users = 50.

inside a specific area) and another or the POST web service (i.e., we simulate an end user submitting a report, which is then stored in the database). We performed the experiments by considering the largest of the synthetically generated data set (i.e., the one with 1,000,000 records) and the most performance (most expensive) configurations of Azure SQL Database and Azure DocumentDB. Specifically, we used the Tier P6 configuration of Azure SQL Database and we set the number of RUs to 60400 for Azure DocumentDB. The achieve results are reported in Figures 1-2.

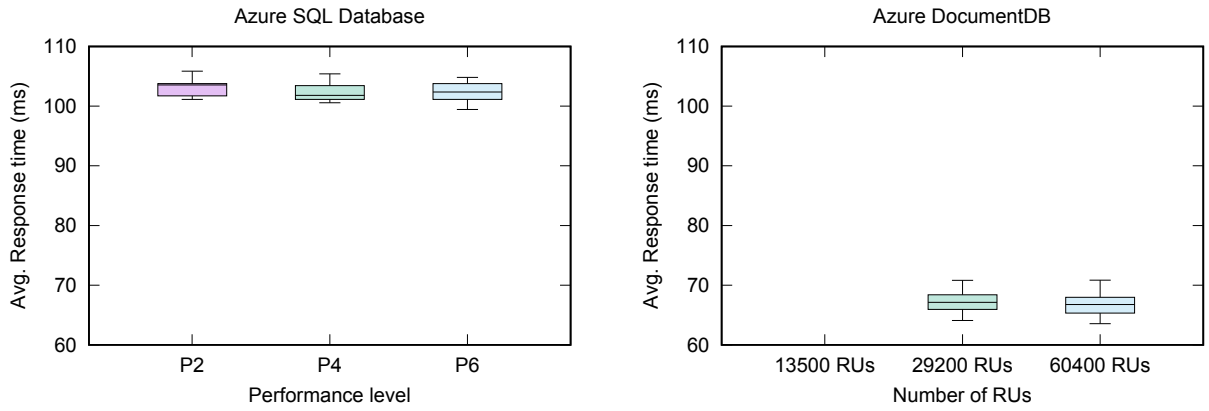
Figure 1(a) shows that the average response time of Azure SQL Database is between 102ms and 103ms for the considered configurations. Hence, on the average, the response time is independent of the number of concurrent users with the considered 1,000,000 records and the Tier P6 version of Azure SQL Database. This means that a good user experience is provided to end users also when 200 users simultaneously use the service. Figure 1(b) reports the results of the analogous experiments when Azure DocumentDB is used to store and retrieve records (reports). Up to 100 concurrent users, the average response time is almost constant (67ms) and Azure DocumentDB is on the average 1.5 times faster than Azure SQL Database. However, when 200 concurrent users are considered, 5% of the requests reach the timeout and the result is not returned. This means that, in some specific time periods of the performed load test, the number of requests, and specifically the related cost in terms of RUs, exceeds the maximum value allowed by the used configuration of Azure DocumentDB. For this reason the average response time when 200 users are considered is not reported in Figure 1(b).[2] Hence, considering the same number of records and the similar performance levels (costs), Azure DocumentDB is faster than Azure SQL Database but it manages less concurrent requests.

We performed a similar set of experiments for the "POST" service. The results, reported in Figures 2(a)-2(b), are consistent with the ones we already discussed. Also in this case, Azure DocumentDB is on the average 1.5 times faster than Azure SQL Database when up to 100 concurrent users are considered but Azure DocumentDB cannot serve all the requests when 200 users are considered. For the POST service, the number of users has also an impact on the response time of Azure SQL Database, that increases when 200 users are considered. Hence, concurrent inserts in Azure SQL Database are less scalable than simultaneous select operations.

Based on the results reported in Figures 1-2, we can also notice that for both databases the average response time of the select operation (GET web service) is at least two times lower than the average response time of the insert operation (POST web service). This is probably related to the fact that select operations can be performed in parallel without needing to lock the input resources.

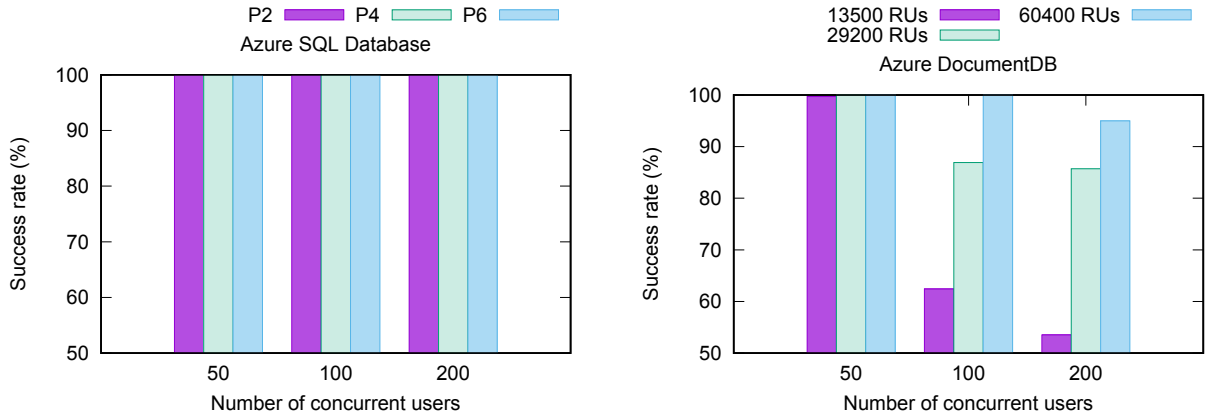
2) Impact of the number of records: Another factor that could potentially impact on the performances of the databases is the number of records already stored in the databases. For this reason we performed a scalability test by varying the number of records from 250,000 to 1,000,000. We considered 50 concurrent users and the most efficient configurations, among the considered ones, of Azure SQL Database and Azure DocumentDB. The results for the GET web service are reported in Figure 3. The achieved results show that the average response time seems to be independent of the database cardinality. In fact, the average response time of Azure SQL Database ranges from 102ms to 104ms (2ms can be considered a non-significant difference considering that the reported response times include also the communication costs). A similar consideration holds for Azure DocumentDB, which is faster than Azure SQL

[2]Since some requests timeout, the average response time returned by the load test in that experiment is not correct because it considers also the execution times of the requests that reached the timeout.



(a) Azure SQL Database. (b) Azure DocumentDB.

Figure 4. GET operation. Average response time with respect to the number of concurrent users. Number of records = 1,000,000. Number of users = 50.



(a) Azure SQL Database. (b) Azure DocumentDB.

Figure 5. Success rate with respect to the performance level of the database. Number of records = 1,000,000.

Database also in this set of experiments. The results obtained for the POST web service, which are not reported in this paper, are similar to the ones discussed for the GET web service.

3) Impact of the database performance level: The last set of experiments is focused on the impact of the performance level of the database. We performed these experiments to understand which performance level, and hence cost, is needed to retrieve and insert data from a geospatial DBaaS database in the considered use case scenario. However, similar considerations holds also for similar applications in analogous geospatial softwares. We performed the experiments by using the 1,000,000 records data set and we set the number of concurrent user to 50.

Figure 4(a) shows that all the considered Azure SQL Database configurations/tiers are characterized by a similar average response time and hence can be used to efficiently and satisfactory answer to the end users requests. Differently, for Azure DocumentDB (see Figure 4(b)) only the top two configurations (characterized by 29200RUs and 60400RUs, respectively) can succeed to answer to the all the concurrent requests of the end users. Hence, for the considered use case, the configuration with 13500RUs is not sufficient and 0.3% of the requests reach the timeout and fail.

To analyze in more details the impact of the performance level of the database on the quality of the application, we analyzed also the success rate[3] of the executed requests with respect to the database configuration and the number of concurrent users. The results reported in Figure 5(a), consistently with the results of the previous experiments, show that Azure SQL Database has always a success rate equal to 100%, independently of the number of concurrent users and the database configuration. Differently, Azure DocumentDB is characterized by some fails (see Figure 5(b)). As expected, the higher the number of concurrent users the lower the success rate. Moreover, the success rate is also clearly related

[3]The success rate is the ratio of the user requests that success.

to the number of RUs and hence to the cost of the considered Azure DocumentDB configuration.

B. Summary of the experimental evaluation

The performed experiments allow drawing some general empirical conclusions about the two geospatial DBaaS-based databases benchmarked in this experiments. Both databases are characterized by good average response times and hence they can be used in the back-end of geospatial applications that select objects based on their location or submit geolocated objects. On the average Azure DocumentDB is faster then Azure SQL Database, also thanks to sharding that allows parallelizing the execution of the user requests over several nodes. However, Azure SQL Database scales better than Azure DocumentDB with respect to the number of concurrent users submitting simultaneous requests.

VI. Conclusion and future work

In this paper, we performed a comparative analysis of geospatial databases for two different point of views: (i) a qualitative comparison based on the features of the considered databases and (ii) an empirical comparison of the average execution times, based on a large campaign of experiments. Both analyses highlight pros and cons of the considered databases and provide some practical insights about the best database depending on the number of expected concurrent users, data set cardinality, and budget.

We plan to extend the performed analysis by considering more complex types of queries, based for instance on intersection between areas, and real data.

Acknowledgment

The research leading to these results has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 700256 ("I-REACT" project).

References

[1] J.-G. Lee and M. Kang, "Geospatial big data: Challenges and opportunities," *Big Data Research*, vol. 2, no. 2, pp. 74 – 81, 2015, visions on Big Data.

[2] Z. Liu, H. Guo, and C. Wang, "Considerations on geospatial big data," vol. 46, no. 1, 2016.

[3] A. Albert, J. Kaur, and M. C. Gonzalez, "Using Convolutional Networks and Satellite Imagery to Identify Patterns in Urban Environments at a Large Scale," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17. New York, NY, USA: ACM, 2017, pp. 1357–1366.

[4] D. Ahlers and E. Wilde, "Report on the Seventh International Workshop on Location and the Web (LocWeb 2017)," *SIGIR Forum*, vol. 51, no. 1, pp. 52–57, Aug. 2017.

[5] N. Bari, G. Mani, and S. Berkovich, "Internet of Things as a Methodological Concept," in *2013 Fourth International Conference on Computing for Geospatial Research and Application*, July 2013, pp. 48–55.

[6] M. Mendoza, B. Poblete, and C. Castillo, "Twitter Under Crisis: Can We Trust What We RT?" in *Proceedings of the First Workshop on Social Media Analytics*, ser. SOMA '10. New York, NY, USA: ACM, 2010, pp. 71–79.

[7] C. Aubrecht, P. Meier, and H. Taubenböck, "Speeding up the clock in remote sensing: identifying the 'black spots' in exposure dynamics by capitalizing on the full spectrum of joint high spatial and temporal resolution," *Natural Hazards*, vol. 86, no. 1, pp. 177–182, Mar 2017.

[8] M. D. McCoy, "Geospatial big data and archaeology: Prospects and problems too great to ignore," *Journal of Archaeological Science*, vol. 84, pp. 74 – 94, 2017, archaeological GIS Today: Persistent Challenges, Pushing Old Boundaries, and Exploring New Horizons.

[9] M. Stonebraker, "SQL Databases V. NoSQL Databases," *Commun. ACM*, vol. 53, no. 4, pp. 10–11, Apr. 2010.

[10] "Microsoft Azure SQL Database." [Online]. Available: https://azure.microsoft.com/en-us/services/sql-database

[11] "PostGIS." [Online]. Available: http://postgis.net

[12] "PostgreSQL." [Online]. Available: https://www.postgresql.org

[13] "Microsoft Azure DocumentDB." [Online]. Available: https://azure.microsoft.com/en-us/services/documentdb

[14] "MongoDB." [Online]. Available: https://www.mongodb.com

[15] "Apache HBase." [Online]. Available: https://hbase.apache.org

[16] "BigTable." [Online]. Available: https://cloud.google.com/bigtable

[17] "Microsoft SQL Server 2016." [Online]. Available: https://www.microsoft.com/en-us/cloud-platform/sql-server

[18] R. Cattell, "Scalable SQL and NoSQL Data Stores," *SIGMOD Rec.*, vol. 39, no. 4, pp. 12–27, May 2011.

[19] W. Lehner and K. U. Sattler, "Database as a service (DBaaS)," in *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, March 2010, pp. 1216–1217.

[20] "OGC Specifications, Simple Feature Access Part 1 - Common Architecture." [Online]. Available: http://www.opengeospatial.org/standards/sfa

[21] "OGC Specifications, Simple Feature Access Part 2 SQL Options." [Online]. Available: http://www.opengeospatial.org/standards/sfs

[22] "ArcGIS." [Online]. Available: http://www.esri.com/software/arcgis

[23] “GeoServer.” [Online]. Available: http://geoserver.org

[24] “GeoTools.” [Online]. Available: http://geotools.org

[25] C. Rossi, M. H. Heyi, and F. Scullino, “A service oriented cloud-based architecture for mobile geolocated emergency services,” *Concurrency and Computation: Practice and Experience*, vol. 29, no. 11, pp. e4051–n/a, 2017, e4051 cpe.4051. [Online]. Available: http://dx.doi.org/10.1002/cpe.4051