Using OGC Services to Interoperate Spatial Data Stored in SQL and NoSQL Databases

Cláudio de Souza Baptista, Odilon Francisco de Lima Junior, Maxwell Guimarães de Oliveira, Fabio Gomes de Andrade, Tiago Eduardo da Silva, Carlos Eduardo Santos Pires

Laboratory of Information Systems – Computer Science Department
Federal University of Campina Grande (UFCG)
Av. Aprígio Veloso 882, Bloco CN, Bairro Universitário – 58.429-140
Campina Grande – PB – Brazil

{baptista, cesp}@dsc.ufcg.edu.br,
{odilonflj, maxmcz, fabiocefetpb, tiagoes}@gmail.com

Abstract. *Spatially-enabled social networks like Twitter and Foursquare have produced huge volumes of geo-referenced information which has been in general stored in NoSQL databases. The need to bring together the entire spectrum of geo-referenced information takes us to the traditional problem of interoperability between SQL and NoSQL spatial databases. This paper proposes a solution for the integration of geographic data stored in both SQL and NoSQL databases using OGC WMS and WFS interoperability services. Experiments conducted on PostgreSQL-PostGIS and CouchDB-GeoCouch spatial databases have demonstrated that it is possible to submit queries using the same syntax for SQL and NoSQL spatial databases in a simple and transparent manner for the user's application.*

1 Introduction

Due to the large volume of data generated on the Internet today, new forms of data storage and data processing are required. We are living in an era of social networks that generate a huge amount of information. For instance, Twitter generates more than 12 Terabytes/day of information which needs to be stored for future reference. Such information can be geocoded. Moreover, the Web 2.0 technology has given rise to several location-based social network services, e.g. Foursquare, Gowalla, Whrrl, Loopt, and Brightkite.

The traditional architectures of Database Management Systems (DBMS) for storing structured data have proven inadequate to deal with this enormous volume of data, known as big data. For these applications, NoSQL databases provide distributed storage and indexing techniques using map/reduce functions [Dean and Ghemawat 2008]. However, the ubiquitous spatial dimension in data sets along with the popularity of spatial applications and also the supporting devices for geo-referenced data gathering, such as smartphones, GPS and cameras, have all contributed to an increase in this

information volume. As a result, some NoSQL databases, such as CouchDB[1] and MongoDB[2], provide support for spatial data.

There is still the pressing need to combine this volume of georeferenced information that emerges from social networks like Twitter and Foursquare with the traditional geo-referenced information, stored, for example, in SQL spatial databases or in spatial data infrastructures. For instance, to view checkin or checkout data from a particular group of users who are shopping within one kilometer buffer of a particular street in the city. This problem involves interoperability between SQL spatial DBMS such as PostgreSQL[3] - PostGIS[4] or Oracle Spatial[5]; and NoSQL spatial database, such as CouchDB - GeoCouch[6] or MongoDB. In other words, we are addressing a problem of geographical data interoperability from highly heterogeneous information sources. At least two strategies may be used to solve this problem. One of them would be to employ a mediator-wrapper architecture [Wiederhold 1992], in which a wrapper would be written to communicate with the NoSQL spatial database, and integrate all database schemas into a common, single relational data schema.

The second strategy would be to implement OGC interoperability services[7] among spatial data, such as Web Map Service (WMS) and Web Feature Service (WFS) on the NoSQL spatial database layer, so as to integrate it to the SQL spatial DBMS by means of a map server; as for instance, GeoServer. This second solution allows any client that implements the WMS and WFS services, for example, the OpenLayers, to submit a query by using the same syntax for SQL and NoSQL spatial databases.

Given these two strategies, we have opted for the second one, which is our main contribution in this paper. To the best of our knowledge, there has been no such NoSQL and SQL spatial database integration using standard OGC web services so far. Consequently, the main contributions of this paper include:

- the implementation of a service layer (OGC WMS and WFS) for the NoSQL CouchDB-GeoCouch database;
- the design and implementation of an architecture to enable interoperability between spatial data stored in SQL and NoSQL databases through OGC services standards; and
- the implementation of a Web map viewer to deploy spatially-aware applications using the proposed interoperable architecture.

[1] The Apache CouchDB Project, http://couchdb.apache.org/
[2] The MongoDB Official Website, http://www.mongodb.org/
[3] The PostgreSQL OpenSource Database, http://www.postgresql.org/
[4] The PostGIS support for geographic objects to PostgreSQL, http://postgis.refractions.net/
[5] The Oracle Spatial, http://www.oracle.com/technetwork/database/options/spatial/overview/introduction/index.html
[6] The GeoCouch – A Spatial index for CouchDB, https://github.com/couchbase/geocouch/
[7] OGC Interoperability Services, http://www.opengeospatial.org/

The remaining of the paper is organized as follows: section 2 discusses related work. Section 3 focuses on the proposed architecture. Section 4 addresses a case study to validate the proposed ideas. Finally, section 5 concludes the paper and points out further work to be undertaken.

2 Related Work

The integration of geographic data stored in different information sources constitutes an old challenge for the geospatial data community; a challenge that has been extensively approached in the literature. An important work was proposed by the project SANY [Havlik et al. 2009]. In this project, a service was designed to provide a single point of access to data spread across the various nodes of a network of sensors. However, this service only supports data provided by the standard Sensor Observation Service[8]. On the other hand, the ORCHESTRA project [Usländer 2007] describes a spatial data infrastructure for risk management applications. The architecture used for its implementation permits the addition of geographic data coming from different sources of information. However, the data must be provided in the form of feature types encapsulated in OGC web services so as to be associated with the infrastructure.

In recent years, the need to process and manage large volumes of data has called for the implementation of effective alternatives to accommodate these tasks. This need has contributed towards the popularization of cloud computing in the geospatial domain. For instance, the map/reduce functions have been used to carry out a number of tasks in the geographical domain, such as the generation of spatial indexes [Akdogan et al. 2010] [Cary et al. 2009], query processing [Jardak et al. 2010], and prediction of natural disasters [Hasenkamp et al. 2010]. However, none of these articles addresses the need for providing the interoperability of their data sets with other existing data.

Moreover, a NoSQL database application to the geospatial domain was proposed by [Miller et al. 2011]. In their work, spatial data are stored in a database implemented in CouchDB. The approach is to use a two-tier architecture for retrieving data from mobile devices. However, in that work there is no interoperability between SQL and NoSQL spatial databases.

The increasing volume of data provided by some geographic data applications has placed a great demand for new ways of storing and managing this kind of information. Lately, these tasks are being addressed via NoSQL databases. Currently, the data offered by this type of database can only be accessed through its native interfaces, which limits access by users and its interoperability with data coming from other platforms. This limitation reinforces the need for a service that allows geographic data to be retrieved using open and standardized interfaces, for which no knowledge of data storage is required.

[8] OGC Sensor Observation Service, http://www.opengeospatial.org/standards/sos/

3 Proposed Architecture

This section describes the architecture used to solve the interoperability problem addressed by this paper. The architecture of our system was developed on three layers: application, service, and persistence. Figure 1 shows the proposed architecture.

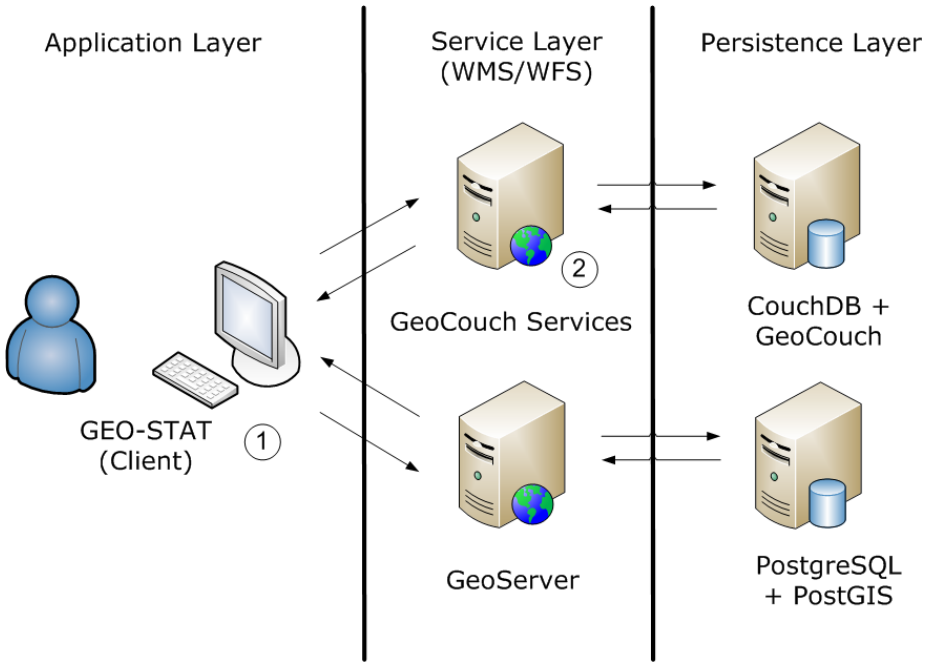


Figure 1: Three-tier architecture for interoperability of spatial data stored in SQL and NoSQL databases.

The application layer is responsible for the interaction between users and services. In our prototype, we used GEO-STAT (**Geo**graphic **S**patio-**T**emporal **A**nalysis **T**ool), a Web map viewer that we have developed (see component 1 of Figure 1). The GEO-STAT tool is based on the Google Maps API, and it works with any server that offers spatial data via WMS and WFS services. This tool provides components to visualize spatial data and spatiotemporal data. It also allows the implementation of spatial queries and the application of spatial filters. In addition, the tool offers an intuitive interface for data mining, based on spatio-temporal clustering and association rules, enabling the visualization of results through map layers. This makes possible, for instance, the implementation of comparative studies between transactional and derived data. Finally, GEO-STAT enables the immediate, practical and intuitive integration and visualization of spatial data available on any publicly accessible server that offers WMS and WFS services.

The service layer defines an interface of how certain features can be accessed by the application layer. Our main contribution lies on this layer, through the **GeoCouchServices** module, a spatial data server that implements WMS and WFS services for the NoSQL GeoCouch database. By means of GeoCouchServices one can

pose queries to a NoSQL database with the same syntax used to query a SQL database in a simple and transparent manner (see Figure 1). The syntax is defined by WMS and WFS standards. It is simple because only the service operations (e.g. GetCapabilities, GetMap, and GetFeature) must be invoked in order to formulate both spatial and non-spatial queries. Transparency to the user is obtained since these services work regardless of the data sources (e.g. GeoServer, Map Server, and GeoCouchServices). It is worthwhile mentioning that the proposed integration between SQL and NoSQL databases is also applicable to non-spatial data.

The GeoCouchServices was developed according to the Model-View-Controller (MVC) architectural pattern. Its purpose is to separate business logic from presentation logic and from application flow control. Figure 2 shows the dependence relationships and the architecture of the GeoCouchServices model (highlighted), component 2 of the architecture, described in Figure 1, for service requests.

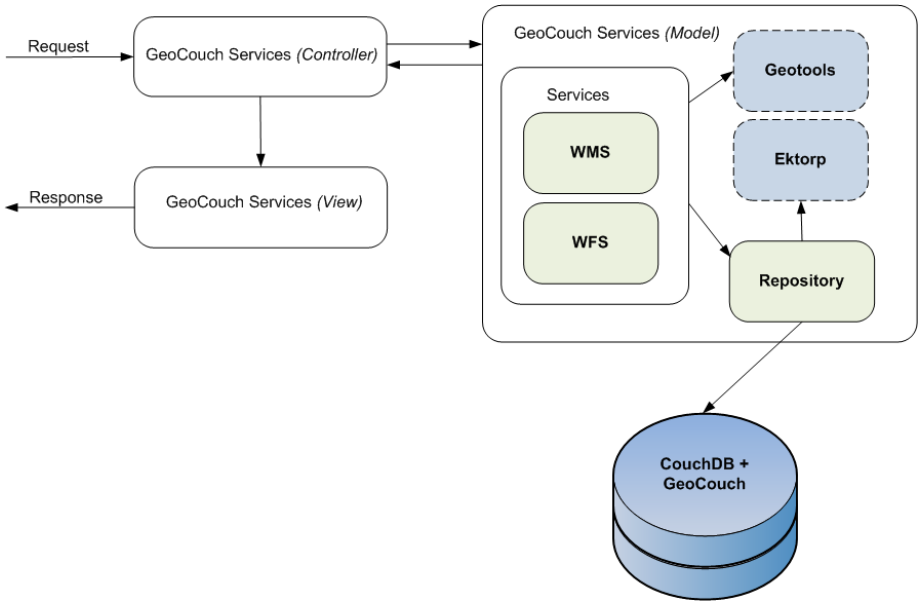


Figure 2: GeoCouchServices MVC architecture.

Upon receiving a request from the application layer, the controller of this service analyzes the request and redirects it to its model responsible for carrying out the request. The service first checks whether the attributes needed to meet the request have been provided. In case a required attribute has not been provided, a service exception is transmitted and a response is generated in the required format. Whatever the outcome, the controller receives a response from the model and forwards the response to the application layer.

For instance, when a GetMap request – including all its mandatory parameters – is submitted, the GeoCouchServices forwards it to the WMS module. The WMS module – via the Repository module – performs a Bounding Box search for the

requested layer in GeoCouch which then returns a file in the GeoJSON format (an open format for encoding a variety of geographic data structures). The Repository module performs a parsing of the GeoJSON file, and transforms it into a collection of features. This collection is then sent to the WMS module which generates the requested map.

Regarding the GeoCouchServices, versions 1.3.0 and 1.1.0 of the WMS and WFS services were implemented, respectively. The implementation of the WMS protocol included only the mandatory operations (GetCapabilities and GetMap). The optional GetFeatureInfo operation is currently being developed. We have also implemented the read-only WFS protocol, including the operations GetCapabilities, DescribeFeatureType and GetFeature.

To implement the WMS and WFS services, we used the GeoTools library [Turton 2008]. The Repository module is a component of the model responsible for forwarding spatial queries to GeoCouch. It is also responsible for querying non-spatial data in CouchDB; through the EKTORP library[9].

At the persistence layer we used CouchDB-GeoCouch for NoSQL data. Despite the possibility of manipulating spatial data in NoSQL with CouchDB and MongoDB, we preferred the former because of the existence of a more complete API that supports most types of existing spatial data.

CouchDB is a document-oriented schema free database. A database is stored as a collection of documents JSON (JavaScript Object Notation), and all interaction is performed entirely by using the HTTP protocol through a RESTful interface [Fielding 2000]. The data are indexed and searched by setting map-reduce views, similar to stored procedures. A view consists of a map function and, optionally, of a reduce function.

In the proposed architecture, we used a spatial extension for CouchDB, called GeoCouch. This architecture stores documents in the GeoJSON format. The GeoJSON[10] emerged as a simple pattern of spatial data format for the Web. This format can represent the following geometric types: point, multipoint, line, multiline, polygon, and multipolygon [Mische 2011].

4 Case Study

This section presents a case study aiming to validate the solution proposed in this paper.

Setup

We configured two servers; each providing WMS and WFS services. The first server used a SQL database; while the second one used a NoSQL database. Both servers stored spatial records about the Brazilian state of Paraíba, including all its 223 municipalities, the highways that cross the state, and all fire outbreaks detected in the state in 2010. All

[9] EKTORP Library – Java API for CouchDB, http://code.google.com/p/ektorp/

[10] GeoJSON – JSON geometry and feature description, http://geojson.org/

records are stored using the WGS84 projection. The records used are real-world data, and were obtained from the Water Management Executive Agency of the State of Paraíba (AESA) and from the National Institute for Space Research (INPE).

For the server that stores a SQL database, we have used the PostgreSQL 8.4 DBMS with a PostGIS spatial extension, version 1.5. In this server, OGC services were made accessible via GeoServer 2.1.0 map server. For the server with a NoSQL database, we have used the CouchDB database with a GeoCouch spatial extension made available by the CouchBase 1.1 package. The OGC services were available from the GeoCouchServices.

Based on this previously established design, we conducted two experiments for the present case study. The goal of the first experiment was to observe the functionality of the OGC requests made to the WMS and WFS services offered by GeoCouchServices. The second experiment evaluated the possibility of interoperability between spatial databases based on SQL and NoSQL.

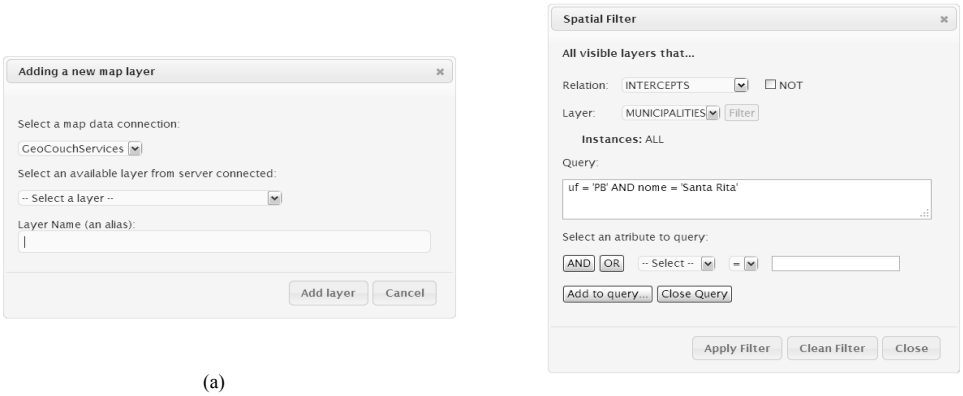
Experiment 1: Checking OGC requests placed to WMS and WFS services

Functional tests were performed on the implemented GeoCouchServices accessing the NoSQL server. The result set was compared to GeoServer in order to validate the accuracy of our implementation. These tests aimed at exploring GetCapabilities and GetMap requests from the WMS service; and GetFeature from WFS, through the use of resources from both servers by means of the GEO-STAT map viewer.

Two servers were configured using the GEO-STAT environment: the server based on GeoServer (SQL), and the other one based on GeoCouchServices (NoSQL). At this stage, for each server, it was only necessary to define an identifying name (alias) for the connection and to provide a way to access the server.

Once the connection configuration through GEO-STAT was established, it was possible to add geospatial layers, and then run the required tests. On selecting a layer to be displayed on the map, the GEO-STAT used the GetCapabilities request (WMS) to return to the list of available layers. Figure 3(a) shows how layers are added using our map viewer. The list of layers available is generated by the application using the response from the GetCapabilities request. Figure 3(b) shows how a spatial filter may be applied to the selected layers. It is possible to spatially filter all added layers in the map (regardless of the source server) by applying a selection filter in to one of them, e.g. filter by municipalities layer where cities with ‘uf’ attribute equals ‘PB’ (Paraíba), and ‘name’ attribute equals ‘Santa Rita’.

Figure 4 shows a map containing the municipalities in the state of Paraíba (223 multipolygons); and the highways that cross the state (959 multilines). These data were obtained using the GetMap request (WMS) sent to the GeoCouchServices.



(a) (b)

Figure 3: Some GEO-STAT forms to: a) add layers; b) apply a spatial filter to the added layers.

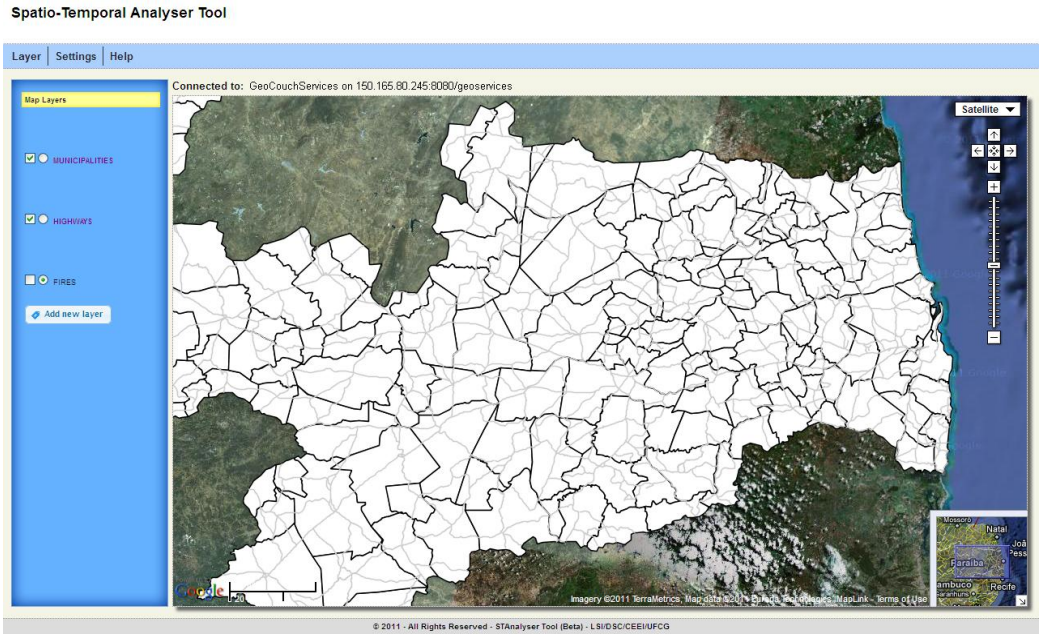


Figure 4: Municipalities and highways in the state of Paraíba provided by the GeoCouchServices using the GEO-STAT map viewer.

The exploitation of the GetFeature (WFS) request was also made possible through the GEO-STAT viewer, which allows us to specify a query using its intuitive graphical interface shown in Figure 3 (b).

Figure 5 shows the result of a query which inquires about all fires that happened in the city of Santa Rita in 2010 (141 points). The data are received by the application as a response of a GetFeature (WFS) request sent to the GeoCouchServices.

To perform this query we first added the layers MUNICIPALITIES and FIRES from GeoCouchServices. Then we applied a spatial filter based on municipalities layer

where the attribute name is equals to 'Santa Rita'. The GEO-STAT viewer, through the GetFeature request, receives all geometry from MUNICIPALITIES corresponding to applied filter and, using again the GetFeature request, uses this received response to request geometries from FIRES that spatially intersects with the geometry of the city of Santa Rita.

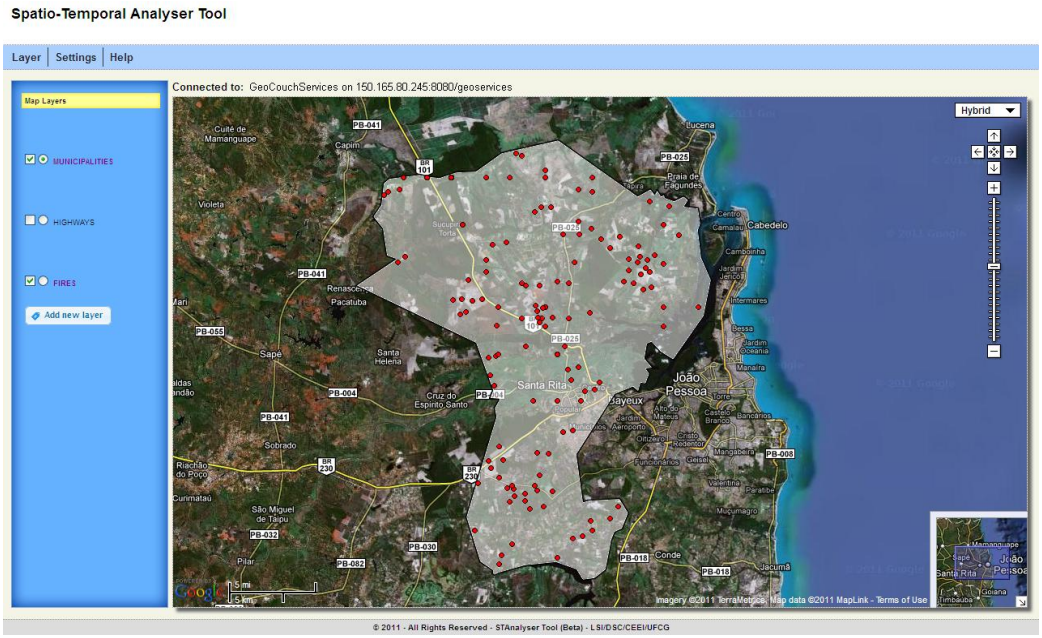


Figure 5: Fires occurred in Santa Rita in 2010 viewed in GEO-STAT, using data provided by the GeoCouchServices.

The comparative tests using the WMS and WFS requests showed that the GeoCouchServices worked satisfactorily, and returned the information similar to GeoServer, as it was expected.

Experiment 2: Evaluation of interoperability between the GeoCouchServices and the GeoServer

To evaluate the interoperability between GeoCouchServices and the GeoServer, i.e., the interoperability between spatial databases based on SQL and NoSQL, we posed queries using WMS and WFS services so that these could access spatial data from both servers.

Since our main goal is to analyze interoperability, we did some modifications on the data stored in the servers. In the first server, based on SQL, we left available only data related to municipalities and highways in the state of Paraiba. In the second server, based on NoSQL, we left available only data about fire outbreaks.

Again we used the GEO-STAT viewer to carry out this assessment. From it, we inserted into the map the MUNICIPALITIES and HIGHWAYS layers provided by the GeoServer. Then we inserted into the same map the FIRES layer made available from the GeoCouchServices. From this point onwards, we made the following spatial query:

Show all fires detected in the city of Monteiro in 2010.

This query may be formulated in the same way as shown in Figure 3 (b). The query is conducted by the GEO-STAT map viewer following two steps. In the first step, the GEO-STAT retrieves along with the GeoServer the geometry and the corresponding identifier of the city of Monteiro in the GML format. Afterwards, the GEO-STAT uses the GetMap (WMS) request to apply the filter in the MUNICIPALITIES layer providing the parameter 'featureid' in the request.

In the second step, the geometry that comes from the first step is used as a filter for a new query sent to the GeoCouchServices, where information is requested on all fires (geometries) that are inside the area represented by that geometry. Table 2 shows the GetFeature request to GeoCouchServices with the geometry of the city of Monteiro retrieved from GeoServer.

Table 2: GetFeature request to GeoCouchServices formed by data from GeoServer.

```
http://150.165.80.245:8080/geoservices/wfs?
   request=GetFeature&version=1.1.0&
   typeName=fires&outputFormat=GML3&
   FILTER=

<Filter xmlns="http://www.opengis.net/ogc"
        xmlns:gml="http://www.opengis.net/gml">
   <Intersects>
      <PropertyName>geometry</PropertyName>
      <gml:MultiSurface srsDimension="2"
                srsName="urn:x-ogc:def:crs:EPSG:4326">
         <gml:surfaceMember>
            <gml:Polygon>
               <gml:exterior>
                  <gml:LinearRing>
                     <gml:posList>
                     -7.94818296 -37.34987508
                     -7.945308   -37.34184996
                     -7.94235897 -37.3172821
                     -7.93552302 -37.31436
                     -7.93376604 -37.30863384
                     ...
                     -7.94818296 -37.34987508
                     </gml:posList>
                  </gml:LinearRing>
               </gml:exterior>
            </gml:Polygon>
         </gml:surfaceMember>
      </gml:MultiSurface>
   </Intersects>
</Filter>
```

The response to this query is received by GEO-STAT in GML format. It contains the geometries and corresponding identifiers (54 points). Then, a new GetMap request with the 'featureid' parameter is sent to GeoCouchServices. The parameter contains all the ids of geometries (fires) separated by commas. The result is shown in Figure 6.

We have successfully implemented interoperability between spatial databases based on SQL and NoSQL in a simple and transparent manner for the user application, satisfying, as a result, our case study and validating the proposed solution.

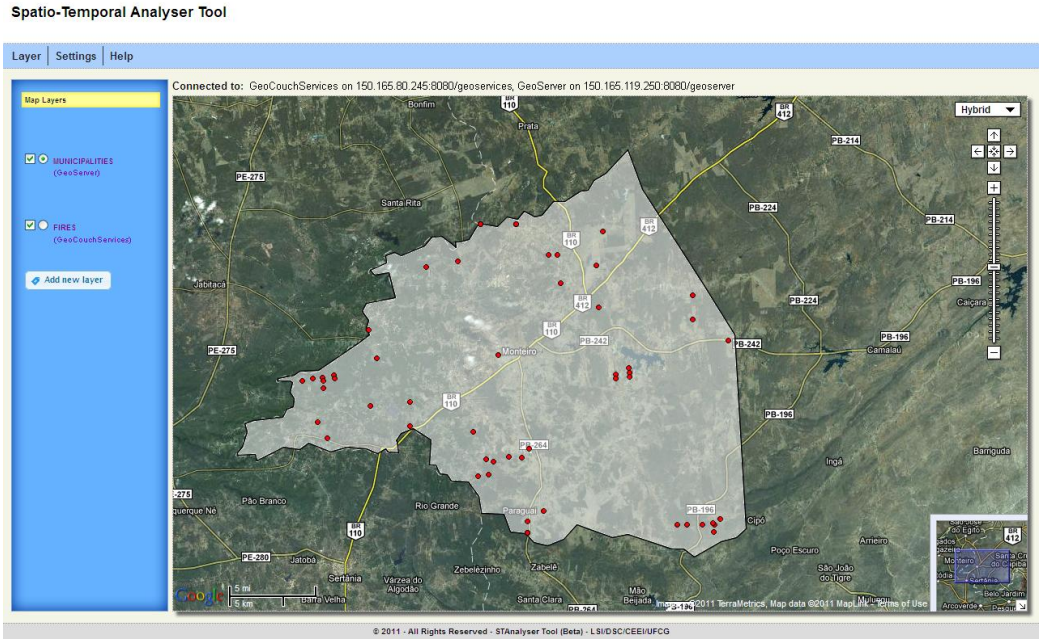


Figure 6: Fires occurred in Monteiro in 2010 using data provided by the GeoCouchServices and the GeoServer.

5 Conclusion and Future Work

This paper proposed a solution that enables interoperability between geographic data stored in SQL and NoSQL databases, using OGC WMS and WFS services.

The functional tests of requests to WMS and WFS services offered by NoSQL server have showed that they work satisfactorily, returning information in much the same way the GeoServer does. Additional tests have demonstrated that it is possible to achieve interoperability between spatial databases based on SQL and NoSQL in a simple and transparent way that will certainly help the user application.

There are at present many ongoing research issues related to the proposed interoperability solution. An objective that will certainly be the focus of our future endeavors will be to conduct experiments on the performance and scalability of services delivered. Another important issue is related to how to add other NoSQL spatial databases such as MongoDB to our architecture.

References

Akdogan, A., Demiryurek, U., Banaei-Kashani, F., and Shahabi, C. (2010). Voronoi-based geospatial query processing with mapreduce. In *Proceedings of the 2010 IEEE*

Second International Conference on Cloud Computing Technology and Science, CLOUDCOM '10, pages 9–16, Washington, DC, USA.

Cary, A., Sun, Z., Hristidis, V., and Rishe, N. (2009). Experiences on processing spatial data with mapreduce. In *Proceedings of the 21st International Conference on Scientific and Statistical Database Management*, SSDBM 2009, pages 302–319, Berlin, Heidelberg. Springer-Verlag.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51:107–113.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. PhD thesis. University of California, Irvine, USA

Hasenkamp, D., Sim, A., Wehner, M., and Wu, K. (2010). Finding tropical cyclones on a cloud computing cluster: Using parallel virtualization for large-scale climate simulation analysis. In *Proceedings of the IEEE 2nd International Conference on Cloud Computing Technology and Science*, CLOUDCOM'10, pages 201–208, Washington, DC, USA.

Havlik, D., Bleier, T., and Schimak, G. (2009). Sharing sensor data with sensors and cascading sensor observation service. *Sensors*, 9(7):5493–5502.

Jardak, C., Riihijärvi, J., Oldewurtel, F., and Mähönen, P. (2010). Parallel processing of data from very large-scale wireless sensor networks. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 787–794, New York, NY, USA. ACM.

Miller, M., Medak, D., and D., O. (2011). Two-tier architecture for web mapping with nosql database couchdb. *Geoinformatics Forum*, pages 62–71.

Mische, V. (2011). CouchDB and GeoCouch. Erlang Factory Lite Munich, http://www.erlang-factory.com/upload/presentations/359/geocouch-online.pdf.

Turton, I. (2008). Open Source Approaches in Spatial Data Handling. Chapter 8: GeoTools. In *Advances in Geographic Information Science* 2, Springer Berlin Heidelberg, pages 153–169.

Usländer, T. (2007). Reference model for the orchestra architecture. Available at: http://portal.opengeospatial.org/files/?artifact_id=20300.

Wiederhold, G. (1992). Mediators in the architecture of future information systems. *Computer*, 25:38–49.