Big Data y Bases de Datos Espaciales: un análisis comparativo

Joana Simões[1][2], Rafael Giménez[1][3] y Marc Planagumà[1][4]

[1] Bdigital, Barcelona Digital centro tecnológico.
[2] jsimoes@ce.bdigital.org.
[3] rgimenez@bdigital.org
[4] mplanaguma@bdigital.org.

El ritmo exponencial de crecimiento en la generación de datos digitales ha convertido la escalabilidad en un factor clave en el diseño de sistemas de información. Aunque esta es un área aún emergente, ya empiezan a existir paquete que permiten extender la infraestructura de Big Data a los datos espaciales. Algunas preguntas que se plantean ante esta situación son: ¿cuándo cambiar las soluciones tradicionales de RDBMS por estas soluciones? ¿Qué configuraciones de arquitectura utilizar? Estas decisiones no sólo se relacionan con el volumen de datos y la velocidad requerida, sino también con los costes económicos asociados al consumo de estos servicios en la nube.

El estudio que se presenta en este documento tiene como objetivo hacer un análisis comparativo entre diferentes soluciones de persistencia y explotación de datos geoespaciales en la nube, basadas en software libre y de código abierto. En concreto, el análisis se centra en la comparación entre el sistema relacional Postgres, ampliado con la extensión espacial PostGIS, y un sistema altamente escalable de almacenamiento y procesamiento basado en clusters, Hadoop. A este ultimo hemos añadido el paquete "Spatial Framework for Hadoop", que permite crear un almacén de datos espaciales sobre MapReduce, y extender la sintaxis nativa de tratamiento de datos (Hive) para permitir gestionar estos datos.

En nuestro análisis comparamos la duración de ejecución de varias operaciones espaciales en los diferentes entornos de prueba: una instancia de Postgres/PostGIS desplegada sobre Amazon Web Services (AWS) y diferentes configuraciones de clusters de Hadoop+Spatial Framework for Hadoop, también desplegada sobre AWS.

Finalmente terminamos con un breve análisis de costes económicos asociados, un factor que puede ser determinante para la adopción de la solución.

Palabras clave: *Base de Datos, SQL, NoSQL, Cluster, Query, Cloud.*

INTRODUCTION

In recent years the increase in scale in traditional data sources due to global changes in business and transportation, plus the explosion in the availability of sensor data, in part caused by the Internet of Things (IoT), resulted in massive volumes of data [1]. Supported by cheaper and widely adopted positioning technologies, a great deal of this data is now geo-located.

Spatial crowd-sourcing movements such as OpenStreetMap[2] or Ushahidi[3], have played an important role in increasing the generation of massive spatial information by the community of users. Being able to digest these large amounts of spatial data in order to understand human behaviour and drive informed decisions, is a current challenge for areas such social sciences or geo-marketing.

According to [4], there are two major requirements for data intensive spatial applications:

- fast query response, which requires a scalable architecture.
- Support to clusters on a cost-effective architecture, such as commodity clusters or cloud environments.

Spatial queries are often computationally intensive [4] since they rely on geometrical operations, not only for computing measurements and generating new objects, but also as logical operations for topology relationships (e.g.: contains, intersects, touches). The increasing volume of spatial information, coupled with the computationally intensive nature of spatial queries demand scalable and efficient solutions.

In this paper we tracked the performance of different spatial warehouse environments on the cloud, regarding a particular set of spatial queries. We have followed a practical approach by focusing on queries that we are currently using, or have been using, rather than queries that may present a specific set of properties, but are not interesting in terms of application. We have also selected datasets that we are currently working with, which although can arguably not be considered "Big Data", do present some performance challenges, and due to its particular nature, will require scalability in the near future. In this way, we hope to provide a useful comparison of SQL and NoSQL environments running on the cloud.

BENCHMARKING SETUP

All benchmarking environments described in this paper were deployed on the cloud, using the Amazon Web Services (AWS) infra structure [5]. The main idea was to compare a centralized Relational Database Management System (RDBMS) with a distributed cluster infra-structure.

Amazon provides the Relational Database Service (RDS), which allows to pick an RDBMS and have it running on a dedicated server, with an optimized configuration. As RDBMS, we have selected PostgreSQL[6], which is widely known and used since 1995. It features its own Free and Open Source License, called PostgreSQL[7]. The *rationale* behind this choice was the ability to use PostGIS[8], a PostgreSQL extension that adds support for geographic objects allowing location queries to be run in SQL. PostGIS has been widely known as one of the most complete spatial extensions for databases, with a very long list of implemented features and a large number of Extract Transform Load (ETL) tools and software, specifically designed to work with it. It is licensed under GPLv2.

Amazon provides different hardware configurations for RDS, according to the price tier. In this benchmarking we used a standard instance from the current generation, which is by no means optimized for heavy work, standing in the bottom end of the scale. The motivation for choosing this configuration was mainly a question of costs. Specifically, we used a db.m3.medium machine, which has a single CPU, 3.75 GB RAM and a 100 GB hard-drive. Amazon describes the performance of this server across the network as “moderate”.

As a distributed system we used Elastic Map Reduce (EMR), an Amazon service that uses Hadoop as framework. Apache Hadoop[9] is a Free and Open Source Software (FOSS), licensed under Apache 2.0[10], for distributed storage and processing in computer clusters. It was specifically designed to manage very large data sets. Hadoop is widely adopted in the “Big Data” world, and it features prominent user such as Facebook or Yahoo! Another reason that drove this choice was the fact that there is a spatial toolkit available for Hadoop[11], provided by ESRI under an Apache License [12]. More specifically we used the spatial framework for Hadoop (SFH), that extends Hive syntax to use a set of geometric functions and types [11].

Amazon offers a very flexible way of setting up the clusters, allowing to choose the number of nodes, and to select the configuration of the master and worker nodes, from a list of fixed hardware configurations. It is only a question of price. Our baseline configuration was a cluster with three nodes, since anything smaller than that does not take much advantage of the parallelism. As a master instance we have selected a general purpose, current generation, machine (m3.xlarge), and as cores we have chosen slightly weaker machines: general purpose, previous generation (m1.medium). Again, in an effort to obey price constraints, we are standing near the bottom end of the scale. To evaluate how the performance changes with the number of machines, we have also created a setup with six nodes, and another one with nine nodes (with similar hardware configurations for master and cores). Finally, to have an idea of the influence of the hardware on the performance, we have created another scenario of a cluster with three nodes, but this time using slightly more powerful instances for the cores. The hardware configuration of the systems used in the benchmarking, is summarized in the table bellow (table 1).

Table 1: Hardware Description

Designation	Description
RDS	db.m3.medium: 100 GB, 1 CPU, 3.75 RAM
EMRx3	master: m3.xlarge; cores: m1.medium (3)
EMRx6	master: m3.xlarge; cores: m1.medium (6)
EMRx9	master: m3.xlarge; cores: m1.medium (3)master: m3.xlarge; cores: m1.medium (3)
EMRx3Large	master: m3.xlarge; cores: m3.xlarge (3)

The exact software versions used in the benchmarking, are summarized bellow (table 2).

Table 2: Software Description

System	Software
RDS	PostgreSQL 9.3.3, PostGIS 2.1
EMR	Hadoop 2.4.0, Hive 0.13.1, GIS tools for Hadoop 2.0

DATASET AND QUERY DESCRIPTION

The list of layers used in the queries is presented in table 3 and figure 1. Since we wanted to perform measurements, all geometry was transformed into a Projected Coordinate System (PCRS), Google Mercator (3857).

Table 3: Layer Description

Name	Description	Geometry	Crs	Other attributes	No of features
tweets_387	Geo-located tweets	point	3857	timestamp, content	18536
grid_10k*	10 km grid enclosing tweets	polygon	3857		10530
points_3857	Bicing stations in Barcelona	point	3857		420
coastline	World coastline	polyline	3857		542216

* PostGIS only

In the following sections we describe each of the four queries used in the benchmarking. Although both Hadoop GIS and PostGIS have a lot of functions that are SQL/MM compliant [13] and therefore similar, the syntax from Hive and Postgres is not exactly the same; thus some queries had to be adapted in order to achieve an equivalent result. For those cases, we present both versions of the query.

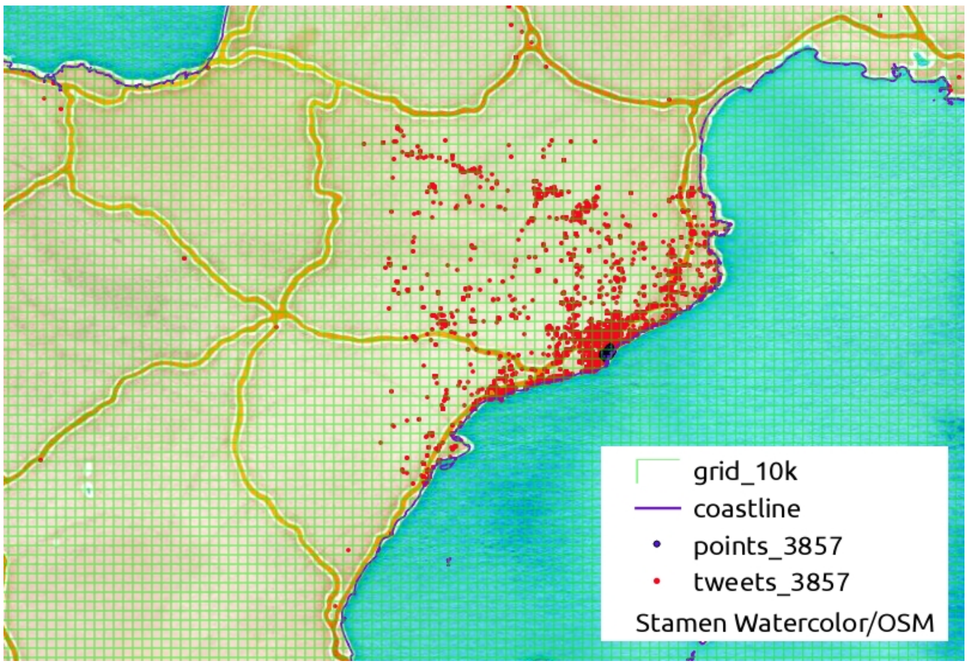

Figure 1: *Layers used in the queries.*

Create Density Grid

Transforming a point cloud into a density surface, is a very common and useful query. It involves defining a grid with a certain resolution, and counting how many points fall within each cell. The results can be stored in a new table.

In this scenario we wanted to create a grid for displaying the density of tweets (grid_cnt). For this particular scale we adopted a grid with a pixel of 10 Km.

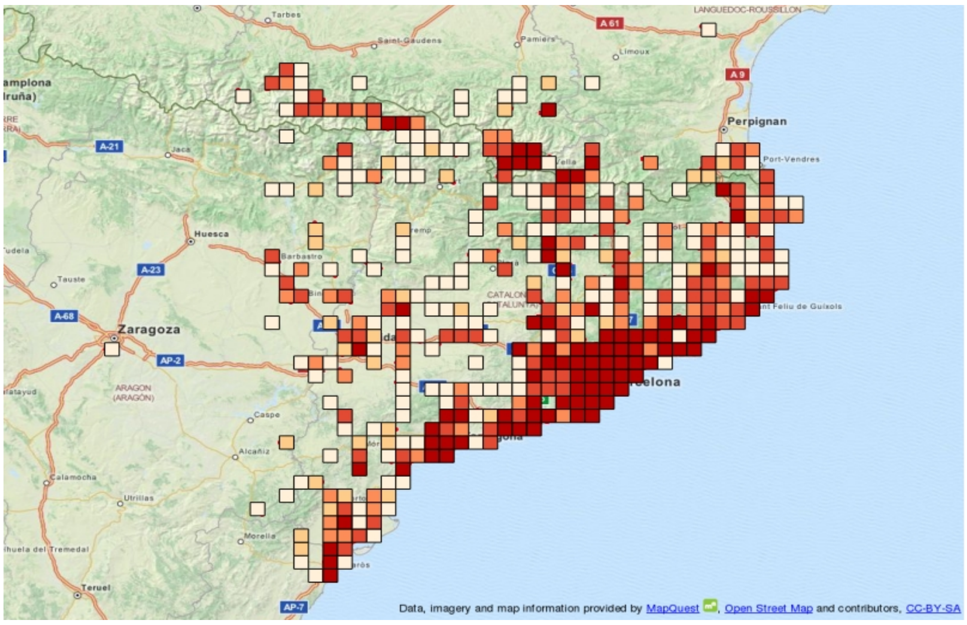
Perpignan
Zaragoza
Huesca
Teruel
Data, imagery and map information provided by MapQuest, Open Street Map and contributors, CC-BY-SA

Figure 2: *Density grid for tweets.*

In PostGIS, this query uses two layers: the tweets layer (tweets_387) and the overlay grid (grid_10k), used to generate the counts.

create table grid_cnt as SELECT grid_10k.id, grid_10k.geom, count(grid_10k.id) as ptcnt FROM grid_10k, tweets_3857 WHERE ST_Contains(grid_10k.geom,tweets_3857.geom) GROUP BY grid_10k.geom,grid_10k.id;

In SFH, it is possible to create the grid as a set of cells (ST_BIN), and fill them with the results (counts) in two-queries.

CREATE TABLE grid_cnt(id bigint, geom binary, count BIGINT);
*FROM (SELECT (ST_Bin(10000, geom)) bin_id, * FROM tweets_3857) bins*
INSERT OVERWRITE TABLE grid_cnt
SELECT bin_id, ST_BinEnvelope(10000, bin_id), count(*) count GROUP BY bin_id;

Buffers

Buffers are another common spatial query. In this scenario we wanted to count how many points lie within 100 m of a bicing station (1079); bicing is a bike sharing system in the city of Barcelona[14].This would give us an idea if people tweet a lot, just before or after they use the bicycle.

This query uses the functions ST_CONTAINS and ST_BUFFER, and is similar in PostGIS and SFH.

select count(*) from tweets_3857 join points_3857 where ST_CONTAINS(ST_BUFFER(points_3857.geom,100),tweets_3857.geom)=true;

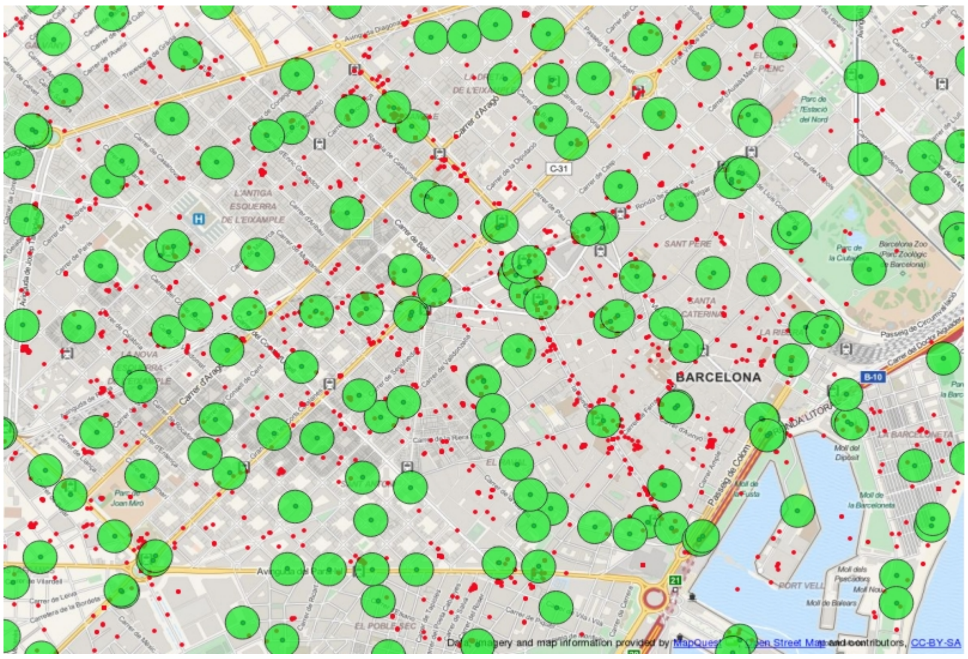
BARCELONA
L'ANTIGA ESQUERRA DE L'EIXAMPLE
SANT PERE
SANTA CATERINA
Parc de la Ciutadella
Barcelona Zoo
Data, imagery and map information provided by MapQuest, Open Street Map and contributors, CC-BY-SA

Figure 3: *Buffers around the 420 bicing stations of Barcelona (detail);*

Select Maximum Distance

Measurements such as areas or distance are another common operation in spatial analysis. In this scenario we wanted to measure the maximum distance between tweets mentioning New Years, on New Years Day (01/01/2015). This query involves a couple of operations; first we filter the dataset for the selected time period and content (hashtag “#2015”); then we calculate the distance between all tweets, and finally select the maximum distance.

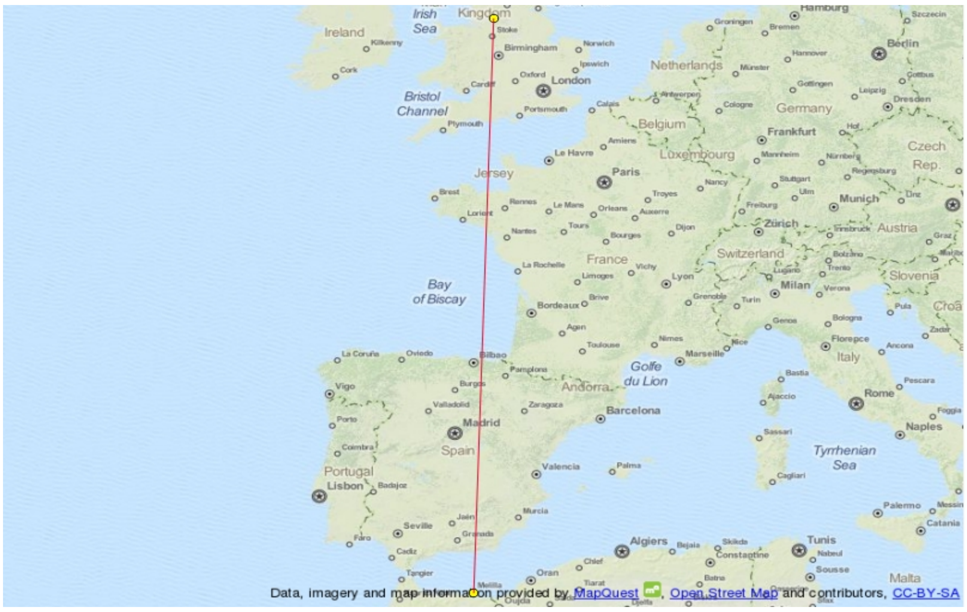

Figure 4: *The most apart tweets regarding New Years, are located in North Africa and the UK;the distance between the two points is 2871.893 Km.*

The key spatial operation in this case is ST_Distance, and the query is exactly the same in PostGIS and SFH:

select max(ST_Distance(a.geom,b.geom)) from tweets_3857 a, tweets_3857 b where a.ts2 > '2014-01-31 00:01:00' and a.ts2 < '2015-01-01 23:59:00' and b.ts2 > '2014-01-31 00:01:00' and b.ts2 < '2015-01-01 23:59:00' and a.content like '%2015%' and b.content like '%2015%';

Import Layer

Importing a layer into the system is a fundamental operation, that happens prior to any other query. In this scenario we wanted to evaluate how long it takes to load a relatively large spatial dataset into the spatial warehouse. The layer is a polyline definition of the world coastline in Well-Known Text (WKT) format, and it has a disk size of 1.5GB.

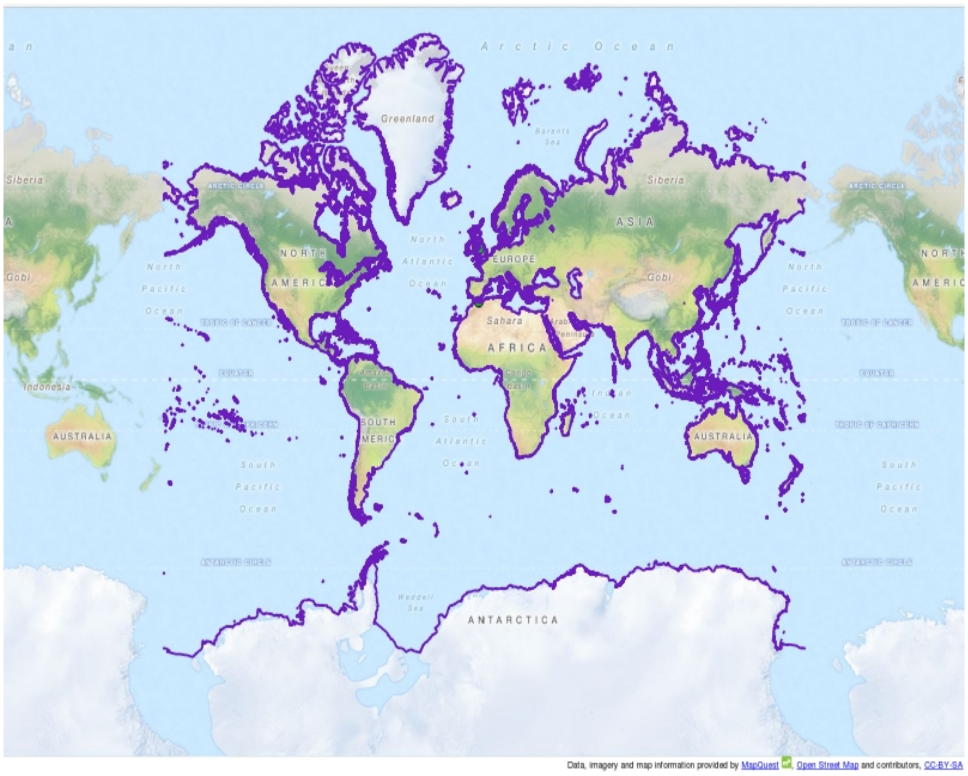

Figure 5: *World coastline.*

The process of importing data from flat files into the system, is relatively different in PostGIS and SFH. In PostGIS we start by creating a table (coastline) to accommodate the text definitions of the geometry; the next step is to fill it from the file, using the \copy directive; then we add a column to store the geometry and update it, instantiating the geometry from WKT (using ST_GeomFromText).

CREATE TABLE coastline(wkt varchar);
\copy coastline from 'coastline.csv' with DELIMITER ' ' CSV QUOTE AS '"' header;
alter table coastline add column geom geometry;
update coastline set geom=ST_GeomFromText(wkt);

To finalize the definition of the spatial table, we would still need to create a spatial index and update the SRID of the geometry.

CREATE INDEX idx_coastline_geom ON coastline USING GIST (geom);
select updategeometrysrid('coastline','geom', 3857);

On the other hand, the import of the layer into SFH, can be performed in two simple steps; first we create an external table that maps the WKT value into a string (from Hive 0.14 it is possible to create a TEMPORARY table instead, which will only live during that session); then we create a new table that instantiates the geometry from that string, at the same time it sets the SRID.

create external table tmp_coastline (wkt string) row format serde
'com.bizo.hive.serde.csv.CSVSerde'
with serdeproperties(
"separatorChar" = "\;",
"quoteChar" = "\"")
stored as textfile LOCATION 's3n://bdigital-benchmarking/coastline/';

create table coastline as select ST_SetSRID(ST_GeomFromText(wkt),3857) as geom from tmp_coastline where WKT NOT LIKE 'WKT';

Note that in both cases the SRID is not optional, since we need it to correctly perform measurements and relate spatial layers.

Since the most expensive query from this set is the one where we update the geometry from WKT, in both systems we focused our benchmarking in this query.

DISCUSSION OF RESULTS

We perform each one of the queries presented in the previous section, in our benchmarking environments: RDS and the different cluster setups discussed on section "Benchmarking setup".

On PostGIS, we run the "explain analyse" function in order to have an idea of what was involved in the query, and the estimated time.

On the clusters we noticed a large variability in query time, that could reach as much as 20 seconds difference. To limit the influence of this variability in the scope of the results, we performed each query 10 times, and used the average value.

On Figure 6, we can see an overview of the results.

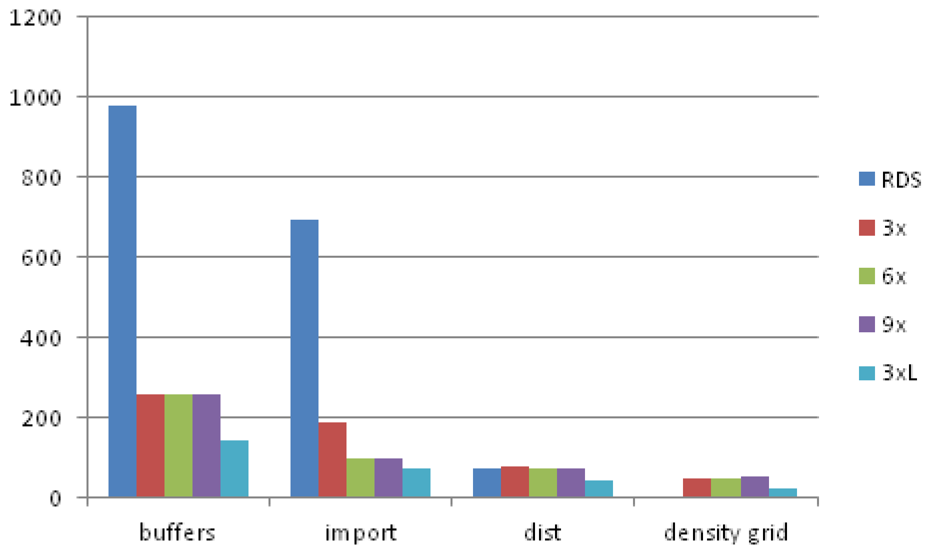

Figure 6: *Results of the benchmarking of the different queries, on different setups on the cloud; query duration is shown in seconds.*

The most expensive query in terms of time, is by far the buffers query. This is also the query where the difference in performance between RDS and the clusters is higher (figure 7). The output of "explain analyse" shows the nested loop involved within ST_CONTAINS as an extremely expensive operation for PostGIS.

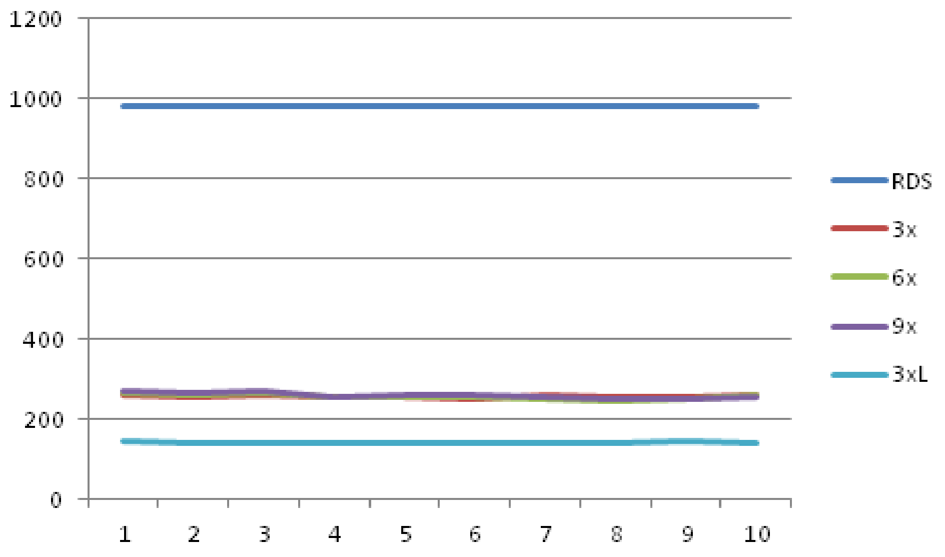

Figure 7: *Average duration of the buffer query.*

The same could be said regarding the import query (figure 8), where the process of importing is quite different in PostGIS and the SFH (see section "Dataset and query description"). Although the best results are also achieved with the cluster with more powerful machines (EMRx3Large), this is the only case where, in average, adding more nodes to the cluster actually results in a gain of performance.

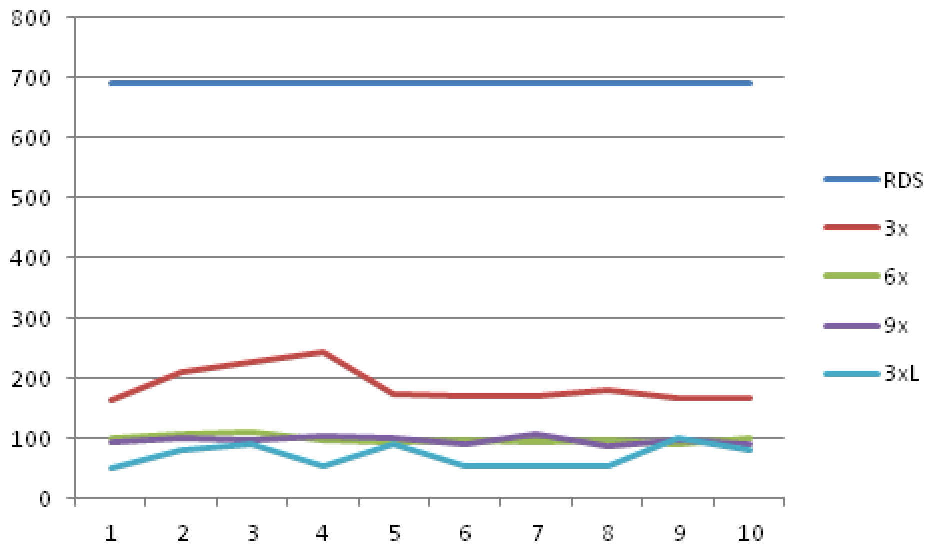

Figure 8: *Average duration of the import query.*

A discussion on Stack Overflow [15] pointed out some different ways of importing large spatial datasets into RDS. The conclusion was that copying features using /copy and update geometry is not the most efficient way; thus for a matter of completeness, we also run an import of an equivalent Shapefile using "shp2pgsql" and the -D directive, a method pointed as more efficient. This is the exact query, that in one go, imports the Shapefile into a new table and sets the SRID:

shp2pgsql -D -s 3857 lines.shp coastline | psql -d benchmarking

This query finishes just under 3 minutes (179 s), being faster than the EMRx3 cluster.

But RDS is not always slower than the clusters. In creating the density grid, RDS is 20 to 50 times faster (figure 8), which could be at least partly explained by the fact that the queries are slightly different in both systems.

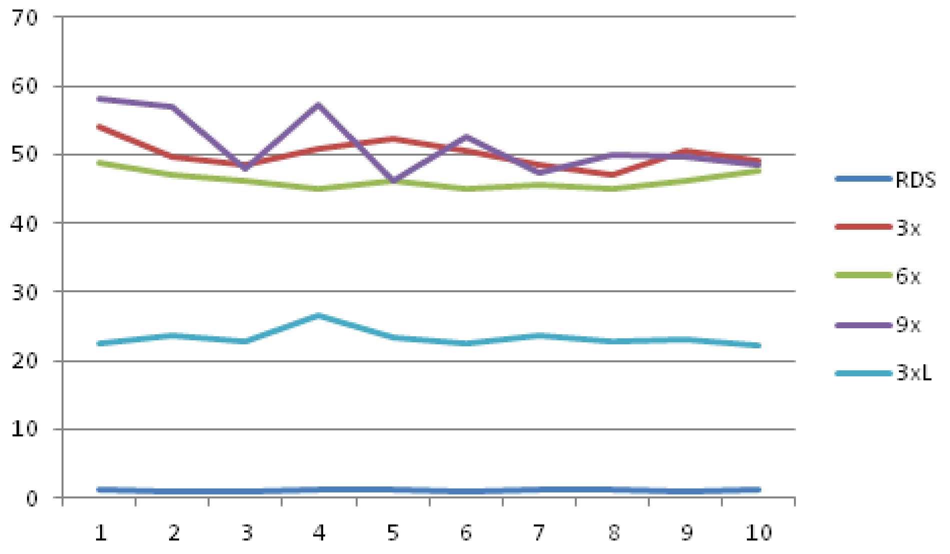

Figure 8: *Average duration of the density grid query.*

The cluster with 3 large machines (EMRx3Large) consistently performs better than the other clusters in every query, and better than RDS in most queries (with the exception of the density grid). Regarding the other 3 clusters (EMRx3, EMRx6 and EMRx9) the results are not so clear. Generally speaking, there are small differences between the query times in these three setups. In three of the queries (density grid, buffers and distance) there is a small gain in using 6 nodes rather than 3, but using 9 workers actually results in larger response times. This could be due to the extra overhead of marshalling the tasks between clusters.

The distance query (figure 9) is the second less expensive query and in this case PostGIS also performs better than the smaller cluster (EMRx3).

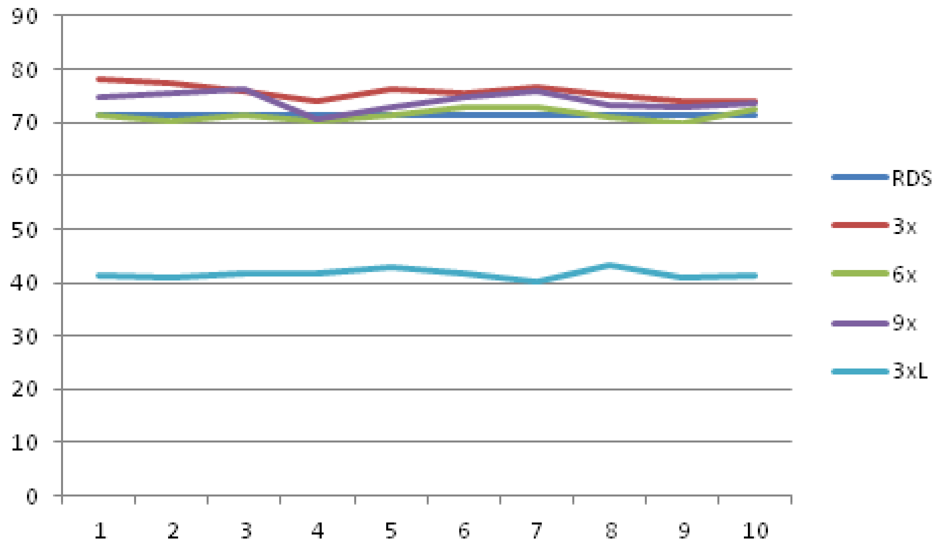

Figure 9: *Average duration of the distance grid query.*

This benchmarking would not be complete without an analysis of the costs. In order to extract some practical lessons from these results and evaluate potential solutions, we need to know how much it costs to setup this infra-structure on the cloud.

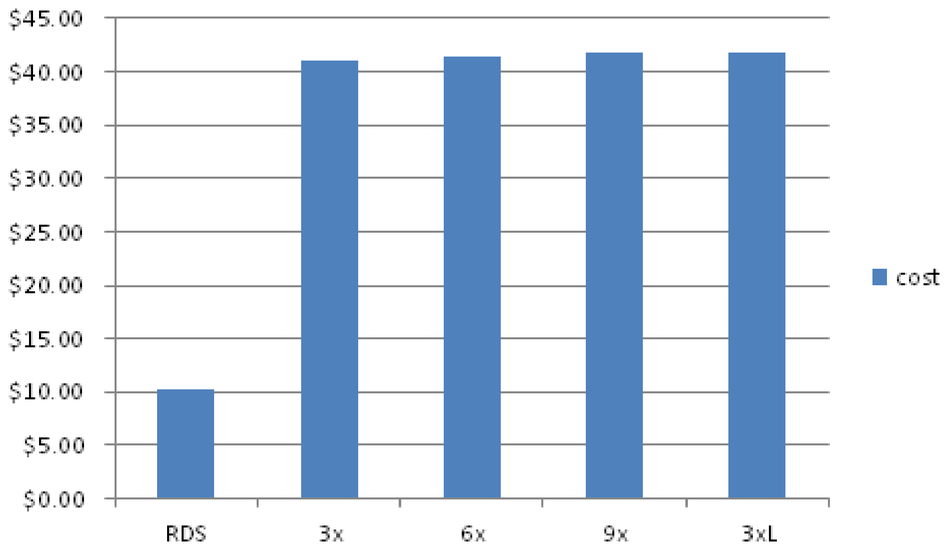

Figure 10: *Cost in dollars, for deploying each benchmarking environment in AWS (1 hour/1month).*

Amazon uses a model that relies on the type of service, machine, and in the case of cluster, number of nodes, to calculate the price. The Amazon Web Services Price Calculator[16] allows us to calculate the exact prices, based on the number of hours per month that we need to rent the infra structure (figure 10). In this benchmarking all the queries could be completed in less than one hour, which means we only need to use one hour of each environment.

CONCLUSIONS

These results have a value within their scope: a working benchmarking, build around the systems we typically use (low cost), the sample sizes that we commonly find (not huge) and the typical operations that we do (measurements, relationship tests, imports). They cannot be easily extrapolated to huge sample sizes, or more powerful systems. Nevertheless relevant lessons, some of them not completely obvious, could be inferred from these experiments.

The first conclusion is that a cluster is not always preferable to a RDBMS. This will actually depend on the query type and we suspect, on the sample size. In theory for an extremely large sample, RDBMS would not work, and much before that the differences in performance would start to be noticed; that is not the case for smaller sample sizes. This idea is confirmed by the fact that RDS performed better in the least expensive queries.

Another conclusion is that adding nodes to a cluster does not necessarily result in a better performance. In the type of query and sample size used in this benchmarking, often smaller clusters outperformed the cluster with more nodes. Adding more nodes to the system increases the overhead in terms of having to distribute the workload; to compensate for that the system has to be more efficient, something that can be only achieved by taking advantage of the parallelism. One possible explanation is that the SFH is not taking full advantage of the MapReduce paradigm; this should be further investigated, since it could result in a bottleneck in an Hadoop-based system. On the other hand having a small cluster (3 nodes) with more powerful machines (EMRx3Large), achieved always a better performance. This gain in performance was more noticeable than the gain in adding more nodes, when it existed (the queries responses for clusters with 3, 6 and 9 nodes were quite similar). This is mostly a gain through vertical scalability, rather than through horizontal scalability, which is what we would normally expect from a cluster.

Regarding prices for one hour usage, RDS is much cheaper (about 4x) than the cheapest cluster presented in the benchmarking. Regarding clusters, the system with 9 nodes (EMRx9) should be left out, as it is more expensive and it generally does not perform better than the ones with 3 and 6 nodes (EMRx3, EMRx6). The system with more powerful nodes (EMRx3Large) has a small price difference, that is largely justified by its gain in terms of performance.

The differences in query syntax are a reminder that Postgres/PostGIS and Hadoop/SFH process things differently, and that alone should force us to evaluate carefully which option is more appropriated for the particular problem we are dealing with.

REFERENCES

- Dunning, T. and Friedman, E. (2014) Time Series Databases: New Ways to Store and Access Data. O'Reilly Media; 1 edition.
- OpenStreetMap. OpenStreetMap. Retrieved March 9, 2015, from: https://www.openstreetmap.org/
- Ushahidi. Ushahidi. Retrieved March 9, 2015, from: http://www.ushahidi.com/
- Aji, A.; Wang, F.; Vo, H.; Lee, R.; Liu, Q.; Zhang, X. & Saltz, J. (2013). Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, 6(11), p1009.
- Amazon. Amazon Web Services. Retrieved March 15, 2015, from: https://aws.amazon.com/
- PostgreSQL. PostgreSQL. Retrieved March 15, 2015, from: http://www.postgresql.org/
- License. PostgreSQL. Retrieved March 9, 2015, from: http://www.postgresql.org/about/licence/
- PostGIS. Spatial and Geographic objects for PostgreSQL. Retrieved March 9, 2015, from: http://postgis.net/
- Apache. Hadoop. Retrieved March 9, 2015, from: https://hadoop.apache.org/
- The Apache Software Foundation. Apache License, Version 2.0. Retrieved March 15, 2015, from: http://www.apache.org/licenses/LICENSE-2.0
- ESRI. Spatial Framework for Hadoop. Retrieved March 15, 2015, from: https://github.com/Esri/spatial-framework-for-hadoop
- ESRI. Spatial Framework for Hadoop license. Retrieved March 15, 2015, from: https://github.com/Esri/spatial-framework-for-hadoop/blob/master/license.txt
- Stolze, K. SQL/MM Spatial: The Standard to Manage Spatial Data in Relational Database Systems. Retrieved March 15, 2015, from: http://doesen0.informatik.uni-leipzig.de/proceedings/paper/68.pdf
- Wikipedia. Bicing. Retrieved March 15, 2015, from: https://en.wikipedia.org/wiki/Bicing
- Stack Overflow. What is the best hack for importing large datasets into PostGIS? Retrieved March 9, 2015, from: http://gis.stackexchange.com/questions/109564/what-is-the-best-hack-for-importing-large-datasets-into-postgis
- Amazon Web Services. Simple Monthly Calculator. Retrieved March 9, 2015, from https://calculator.s3.amazonaws.com/index.html