

Evaluating a NoSQL alternative for Chilean Virtual Observatory Services

Jonathan Antognini, Mauricio Araya, Mauricio Solar, Camilo Valenzuela, and Francisco Lira

Universidad Técnica Federico Santa María

Abstract. Currently, the standards and protocols for data access in the Virtual Observatory architecture (DAL) are generally implemented with relational databases based on SQL. In particular, the Astronomical Data Query Language (ADQL), language used by IVOA to represent queries to VO services, was created to satisfy the different data access protocols, such as Simple Cone Search. ADQL is based in SQL92, and has extra functionality implemented using PgSphere. An emergent alternative to SQL are the so called NoSQL databases, which can be classified in several categories such as Column, Document, Key-Value, Graph, Object, etc.; each one recommended for different scenarios. Within their notable characteristics we can find: schema-free, easy replication support, simple API, Big Data, etc. The Chilean Virtual Observatory (ChiVO) is developing a functional prototype based on the IVOA architecture, with the following relevant factors: Performance, Scalability, Flexibility, Complexity, and Functionality. Currently, it's very difficult to compare these factors, due to a lack of alternatives. The objective of this paper is to compare NoSQL alternatives with SQL through the implementation of a Web API REST that satisfies ChiVO's needs: a SESAME-style name resolver for the data from ALMA. Therefore, we propose a test scenario by configuring a NoSQL database with data from different sources and evaluating the feasibility of creating a Simple Cone Search service and its performance. This comparison will allow to pave the way for the application of Big Data databases in the Virtual Observatory.

1. Introduction

One of the requirements of the current ChiVO functional prototype (Antognini et al. (2014)), is the development of our own *Name Resolver* for astronomical objects that will be indexed by Alma. First we studied how other VO web services implement their name resolvers, and unsurprisingly, almost all of them use the CDS Sesame Name Resolver (Wenger et al. (2000)). Sesame uses SIMBAD data for resolving its queries, and send the results back to the user in the form of a XML file. SIMBAD itself offers a variety of search methods, one of them is *Coordinate Query*, similar to IVOAs Simple Cone Search standard (Williams et al. (2006)).

To have a similar database to SIMBAD, we have implemented two query methods that SIMBAD offers: a name resolving (LIKE operand query) and ConeSearch. For the second one, IVOA recommends using Postgresql along with Pgsphere module. However, we wanted to take a look in the new generation of databases engines, namely NoSQL databases, and compare the performance of one of them against Postgresql using Pgsphere.

2. Why NoSQL database

The huge amount of new data that ALMA observations will produce, obliges handling data more efficiently, so we could use the new technologies in our advantage starting with how to data is stored for the variety of IVOA compliant services in ChiVO.

To understand the differences between a SQL and NoSQL databases engines, we have to recall that the first one is based on ACID principle (Atomicity, Consistency, Isolation and Durability) to ensure the transactions are made correctly, but those same properties makes SQL databases weak in horizontal scaling. On the other hand, NoSQL databases are based on the CAP principle (Consistency, Availability, Partition Tolerance), which makes them really useful for distributed environments, but they doesn't support complex transactions or joins.

To choose which NoSQL database engine could be useful for ChiVO, we base our decision on the fact that almost every IVOA Data Access Layer Protocol (DAL) need performing some type of spatial search. We have considered the Redis key-value database, due to its simplistic way to store data and ease to use, but it was very limited in their queries. Neo4j is a really complex graph database, which might help to understand the relationships between data, but it is hard to manipulate and slower than other databases. Cassandra, a column database, its similar to SQL databases, but it has no support for spatial indexes. At last, Mongo document database, has a built-in geospatial module, doesn't need a schema, and has reliable and thoughtful of documentation.

For our tests we've chosen MongoDB, not only for the documentation, but also for the variety of query methods that it offers and the support of spatial indexes.

3. MongoDB versus Postgresql performance

We prepared some tests for Mongo to compare it with Postgresql. From the user point of view, we tested *read* capabilities of both databases engines, and for the developers points of view, we measured the *write* operation performances. The tests were made in the same machine with Ubuntu 13.04.

We started by testing both databases engines write capabilities, using a collection of 10.000 entries with data from SIMBAD database, and inserted them into the databases. We parsed the data into a JSON and a SQL file, inserting data row by row into each of the databases, and measured the time until we finished populating the database. The result was slightly superior performance of MongoDB with respect to PostgreSQL. (*see table1.*)

	MongoDB	Postgresql
10.000 entries insertion	3.12468409538 s	3.84547805786 sec

Table 1. LIKE operand run-time comparison, using Regular expression in MongoDB; with Postgresql 8.4, and MongoDB 2.6.4

With both databases populated, we were ready to test the read capabilities tests. The first read test was the SQL LIKE operator, because is particularly useful for the name resolver application. MongoDB doesn't have LIKE operator, but we can use regular expressions to produce a similar query. Unfortunately, MongoDB was 7 times slower than PostgreSQL to make the query.(*see table2.*)

	MongoDB	Postgresql
LIKE operator query	15.5ms	2.79 ms

Table 2. LIKE operand run-time comparison, using Regular expression in MongoDB; with Postgresql 8.4, and MongoDB 2.6.4

The second read test was to measure how precise is MongoDB spatial queries compared to the same query in Postgresql. The result was the same amount of rows in all the 7776 queries made.

At last, we tested the speed of spatial search, vital for conesearch and other DAL protocols, by making the same query with MongoDB and Postgresql. We covered the northern celestial sky using a fixed radius, and compute the mean of the query's run-time. Then, we repeated the procedure with different radius values. MongoDB uses less than 0.1 ms in small queries, but when we use a large radius, the time used in the query started to grow fast.(see table3.)

Radius	MongoDB	Postgresql
2 arcsec	0.0594135802 ms	0.04938734567 ms
2 arcmin	0.0652336081 ms	0.04964043209 ms
5 degree	0.5216049382 ms	0.18964429012 ms
20 degree	8.7623456790 ms	2.05688194444 ms
35 degree	33.432870370 ms	7.83297376543 ms
50 degree	55.912037037 ms	15.3577060185 ms

Table 3. Conesearch query run-time comparison, using the same radius and changing the center of the cone

4. No-SQL Prototype Service Implementation

After the tests, we implemented a functional prototype of a SIMBAD style database, with its own Conesearch and Name Resolver functionalities inside ChiVO prototype. It has been implemented inside the Endpoint layer of ChiVO. (see fig1.)

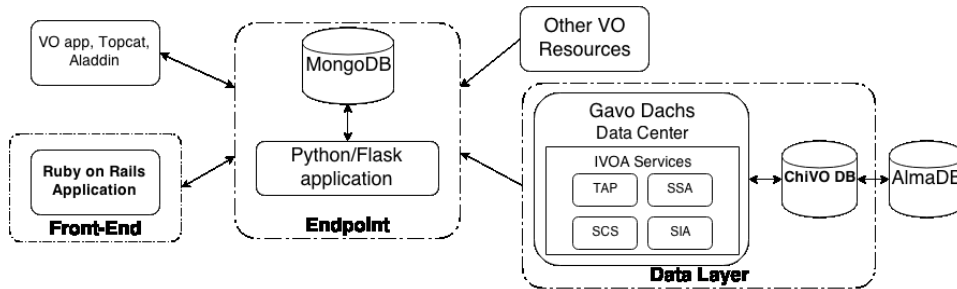


Figure 1. ChiVO Webservices Architecture

Using MongoDB and Astropy Python library, we implemented a Conesearch service for our own astronomical database, we receive the queries, and transform the search radius from degrees to radians to send it over to MongoDB, then we return the

data in form of a VoTable file. The prototype is available in [http://dachs.lirae.cl/bib/conesearch?RA=\(right_ascension\)&DEC=\(declination\)&SR=\(radius\)](http://dachs.lirae.cl/bib/conesearch?RA=(right_ascension)&DEC=(declination)&SR=(radius)).

For the Name Resolver part of the functionalities, we made a query using regular expressions to MongoDB and then return the position of the object found in form of a XML, and it is available in [http://dachs.lirae.cl/name_resolver/\(name\)](http://dachs.lirae.cl/name_resolver/(name)).

5. Conclusion

Implementing a Conesearch using new technologies has proved to be really simple with MongoDB, because it has proper documentation, and a growing community, in contrast of implementing it with PostgreSQL and PgSphere module. Moreover, PgSphere is getting deprecated, has nearly no active community, and doesn't work with the newer versions of PostgreSQL. However, MongoDB lacks performance in a single server, but its a viable alternative for the implementation of a DAL service in a distributed environment because of his good horizontal scalability.

References

- Antognini, J., Solar, M., Ibsen, J., Araya, M., Nyman, L., Mardones, D., Valenzuela, C., Ramirez, P., Fernandez, C., & Garces, M. 2014, in SPIE Astronomical Telescopes+ Instrumentation (International Society for Optics and Photonics), 91522P
- Wenger, M., Ochsenbein, F., Egret, D., Dubois, P., Bonnarel, F., Borde, S., Genova, F., Jasiewicz, G., Laloë, S., Lesteven, S., et al. 2000
- Williams, R., Hanisch, R., & Szalay, A. 2006