Volume Title
ASP Conference Series, Vol. **Volume Number**
Author

Evaluating a NoSQL alternative for Chilean Virtual Observatory Services

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

Universidad Tcnica Federico Santa Mara

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

We have been working in the develop of a functional prototype for ChiVO, one of the requirements was to have our own Name Resolver for astronomical objects that will be indexed with ALMA data. First we looked how the other VOs web services implement their name resolvers, and almost all of them use CDSs Sesame Name Resolver; Sesame uses SIMBAD data for their queries, and send the results back to the user, and SIMBAD itself offers a variety of search methods, one of them is Coordinate Query, similar to IVOAs Simple Cone Search standard.

To have a database similar to SIMBAD, we started implementing two query methods that SIMBAD offers, a simple criteria search, and ConeSearch; for the second one, IVOA recommends using Postgresql along with Pgsphere module. But Pgsphere last version doesn't benefits with the new versions of Postgresql, so we look for the new generation of databases, No-SQL databases, and compare the performance of one of them against Postgresql with Pgsphere.

	MongoDB	Postgresql
10.000 entries insertion	3.12468409538 s	3.84547805786 sec

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

2. Why No-SQL database

The huge amount of new data that will come from ALMAs observations, will make us have to handle a big amount of data, so we could use the new technologies in our advantage, starting with how to store the data, for the variety of IVOA compliant services from ChiVO.

First we take a look what are the differences between a SQL and No-SQL databases, the first one is based on ACID (Atomicity, Consistency, Isolation and Durability) to ensure the transactions are made correctly, but that same properties makes SQL databases weak in horizontal scaling; No-SQL databases are based on the CAP theorem (Consistency, Availability, Partition Tolerance), and makes them really nice for distributed systems, but they doesn't support complex transactions or joins.

To choose which No-SQL database could be useful for ChiVO, we base our decision in the spatial search needed for almost every IVOAs Data Access Layer Protocol (DAL); we take a look in Redis key-value database, its simplistic way to store data make it easy to use, but it was very limited in their queries; Neo4j, a graph database is a really complex database, helps to know relations between data, but make it hard to manipulate and slower than other databases; Cassandra, column database, its similar to SQL databases, and has his own query language; Mongo document database, has a built-in geospatial module, doesnt need a schema, and has lots of documentation.

For our tests we choose MongoDB, for his huge amount of documentation, his variety of query methods, and their spatial indexes.

3. MongoDB versus Postgresql performance

We prepare some tests for Mongo to compare it with Postgresql, for the user point of view, we tested read capabilities of both databases and for the developers points of view, we look in write operation performance.

We started testing both databases write capabilities, using a collection of 10.000 entries with data from some votables, to insert them in the databases, we parsed the data into a JSON and a SQL file for each database, then insert the data row by row into the databases, and took the time until we finished poblating the database, and we got a slightly difference, but MongoDB was a little faster than PostgreSQL. (*see fig 1*)

After we got the data in both databases we started with the read capabilities tests.

The first test was with SQLs LIKE operator, this will be extremely useful for the name resolver part of the database. MongoDB doesnt have LIKE operator, but we can use regular expression to have something similar. MongoDB took 7 times more time than PostgreSQL to make the query.(*see fig2.*)

The second test was to measure how precise was MongoDB spatial queries comparing it with the same query in Postgresql. giving us the same amount of rows in all the 7776 queries made.

	MongoDB	Postgresql
LIKE operator query	15.5ms	2.79225 ms

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

Radius	MongoDB	Postgresql
2 arcsec	0.0 ms	0.049387345679 ms
2 arcmin	0.0 ms	0.0496404320988 ms
5 degree	0.521604938272 ms	0.189644290123 ms
20 degree	8.76234567901 ms	2.05688194444 ms
35 degree	33.4328703704 ms	7.83297376543 ms
50 degree	55.912037037 ms	15.3577060185 ms

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

At last we test the speed of spatial search, vital for conesearch and other DAL protocols, making the same query with MongoDB and Postgresql, we covered the northern celestial sky using a fixed radius, we take the mean of the query's run-time, and repeating the procedure with different radius. MongoDB uses less than 0.1 ms in small queries, but when we use a big radius, the time used in the query started to grow fast.(*see fig3.*)

4. ChiVo's Name Resolver Implementation

1. **The database.** Chivo's Name resolver uses a Mongo database, with the built-in 2sphere indexes to make the Conesearch queries.

2. **Architecture.** The Chivo architecture is composed of the Front-end web application, a Endpoint web application, and a Data layer (see fig). The Front-end is a Ruby on Rails application, that get the data from the Endpoint layer. The Data layer has is a GAVO Dachs data center, and has some of the DAL protocols implemented. At last the Endpoint layer communicates the Front-end and the external applications with our Data layer and serve as a Load Balancer. (*see fig 4*)

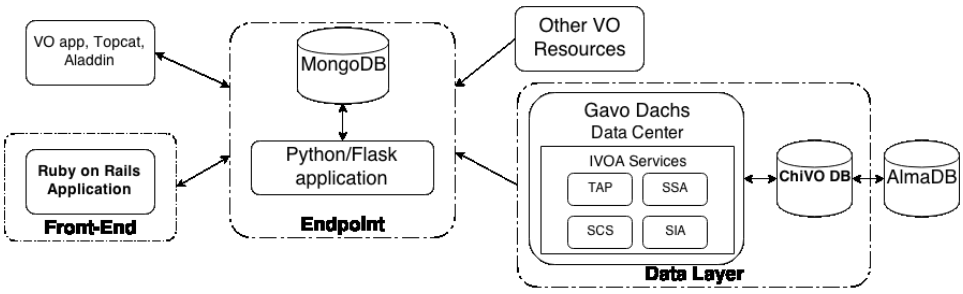

Figure 4. ChiVO Webservices Architecture

3. **The Endpoints web interface.** ChiVO Endpoint uses Flask, to have a lightweight application; Flask is a Microframework written in python, we also can handle the data with astropy libriaries, to make the mongo query and parse the results to VoTable so we can fulfill IVOA Conesearch standard.

4. **Name Resolver Interface.** We handle the request in the Flask Endpoint, then make a query to the MongoDB database containing the object name, at the end we return the user the coordinates of the object in a JSON format. The implemented prototype is available in `http://dachs.lirae.cl/name_resolver/(name)` for the name resolver and in `http://dachs.lirae.cl/bib/conesearch?RA=(right_ascension)&DEC=(declination)&SR=(radius)`.

5. What's next

Implementing a Conesearch using new technologies has proved to be really simple with MongoDB, because it has lots of documentation, and a growing community, in contrast of implementing it with PostgreSQL and PgSphere module, PgSphere is getting deprecated, has nearly no active community, and doesnt work with the newer versions of PostgreSQL. But MongoDB lacks performance in a single server, makes it not a good choise to change from PostgreSQL to MongoDB.

The next step is start looking how to update PgSphere to use the new features of PostgreSQL 9.3+, like JSON datatype and replication improvements. Test MongoDB clusters read capabilities, develop indexation for column based databases, that offer better read capabilities than document based databases.