Universidad de Granada / Instituto de Astrofísica de Andalucía

Integrating NoSQL Technologies into the Virtual Observatory to Support Big Data Challenges

Máster en Métodos y Técnicas Avanzadas en Física
Trabajo Fin de Máster presentado por José Antonio Magro Cortés
y supervisado por el Dr. Juan de Dios Santander Vela

2013

Contents

Acknowledgments

En primer lugar debo reconocer la deuda que tengo con mi tutor, el Dr. Juande Santander, por aceptarme como alumno, por compartir sus conocimientos conmigo, por sus revisiones, anotaciones y correcciones. Por su disponibilidad a pesar de la distancia, de la diferencia horaria y del resto de circunstancias. Por la pasión que le pone a su trabajo. ¡Gracias maestro!

A la Dra. Maribel Tercedor, porque sin ella preverlo, hace 15 años una frase suya cambió mi rumbo laboral. Este bandazo académico se lo debo a ella.

A Mamen, Antonio y Alejandro por permitir embarcarme en esta locura a pesar de las horas que les iba a robar y por aportar la cordura necesaria para compatibilizar trabajo, estudios y familia.

A mis padres, por contagiarme el virus de la curiosidad a base de comprarme libros y más libros y darme toda clase de facilidades.

A mis abuelos: el éxito de vuestra lucha se refleja en momentos como éste, a pesar de que últimamente algunos se empeñan en destruir vuestro legado. No lo conseguirán.

A Laura y María del Mar, compañeras, por las risas entre clase y clase (y fuera de ellas) y por haber ayudado a este intruso*. Llegaréis lejos, os lo merecéis.*

List of Figures

List of Tables

Listings

Chapter 1

Introduction

The volume of data produced at science centers presents a processing challenge that is getting more and more difficult to deal with. At the European Organization for Nuclear Research[1] (CERN), for instance, the Large Hadron Collider[2] (LHC) will record several hundred million collision events, which are recorded so that physicists can match them against simulated events (again, millions of them), and then determine if the collisions have thrown up any interesting information.

CERN, like many other science centers, does not have the computing or financial resources to manage all of the data on site, so it moved into the grid to share the load with computing centers around the world. The Worldwide LHC Computing Grid is a distributed system which gives over 8000 physicists near real-time access to LHC data.

Other current or future astronomical facilities such as the Low-Frequency Array[3] (LOFAR), the Australian Square Kilometre Array Pathfinder[4] (ASKAP), the Large Synoptic Survey Telescope[5] (LSST), or the Square Kilometre Array[6] (SKA), will also be delivering much more data that it is feasible for the community to directly download or process. Tools for making it easier for astronomers to operate on those larger datasets are needed, and indeed are priorities of those facilities, as we are moving to a data-intensive paradigm [**?**], in which large

[1] http://home.web.cern.ch

[2] http://home.web.cern.ch/about/accelerators/large-hadron-collider

[3] http://www.lofar.org

[4] http://www.atnf.csiro.au/projects/mira/

[5] http://www.lsst.org/lsst/

[6] http://www.skatelescope.org/

datasets are driving the way we derive meaning in all scientific fields.

However, current facilities are also suffering from the lack of dedicated tools for astronomers, as their data and metadata sizes are also increasing beyond what can be comfortably manipulated and downloaded from the personal workstations of scientists. For instance, the datasets from the Atacama Large Millimetre and Submillimetre Array[7] (ALMA), the largest radio interferometer currently in operations, typically occupy several gigabytes in their current configuration, and will become larger as more and more antennas and receivers are added, increasing the telescope sensibility and the number of available baselines.

In order for observatories like ALMA to store and publish their observations, they have typically relied on relational database management systems (RDBMS; see [**?**]). However, RDBMS have their weaknesses, specially for data publishing applications, as they impose the same schema for datasets which might not have all of their metadata in common, and mandate an ingestion phase that converts the metadata in the original format into a format suitable for RDBMS systems.

Today, non-relational, cloud, or the so-called NoSQL (*Not-only-SQL*) database systems are growing rapidly as an alternative model for database management. These systems are tuned for the kind of big data applications that have made possible very large systems such as Google or Facebook, and can incorporate the documents themselves, and query on the existing metadata, without the need for a dedicated, complicated ingestion phase.

In this work, we give a short overview of the Big Data challenges being faced by astronomy, and present an alternative, using one of the freely available NoSQL databases, and how it can be integrated in the Virtual Observatory framework.

[7] http://www.almaobservatory.org/

Chapter 2

Big Data challenges in astronomy

Astronomical datasets are growing at an exponential rate: the next generation of telescopes will collect data at rates of several terabytes per day. This data deluge presents some critical challenges for the way astronomers can get new knowledge from their data. These extremely large datasets, or datasets with high data rates, are commonly known as *Big Data*.

Big Data are high-volume, high-velocity, and/or high-variety information assets that require new forms of processing to enable enhanced decision making[1], insight discovery and process optimization. Examples of Big Data outside of astronomy (see, for instance, the Nature Special on Big Data issues [**?**]) include information systems such as Web search results, electronic messages (e.g. SMS, email and instant messages), social media postings, pictures, videos, or large web system log data. These situations require the processing of terabytes and even petabytes of data. This is achieved by means of distributed, segmented processing.

Relational Database Systems (RDBMS) are found to be inadequate in distributed processing involving very large number of servers and handling Big Data applications, due to the limitations of query speed, storage size, and scalability inherent to the tasks of keeping relationships between database tables, atomicity of transactions, and the storage layout of data in disk.

In the next sections, we will review some of the current and future challenges for astronomical data processing, which justify the search for NoSQL solutions for astronomical data sharing.

[1] For instance, fast follow up of transients in the data collected by the LSST.

Figure 2.1: Artist's impression of ALMA dishes at the Chajnantor plateau, in the Atacama Desert

2.1 The Atacama Large Millimetre/Submillimetre Array

The Atacama Large Millimetre/Submillimetre Array (ALMA) is a worldwide project, built cooperatively by the European Southern Observatory (ESO), the National Radio Astronomical Observatory (NRAO) of the United States of America, and the National Astronomical Observatory of Japan (NJAO), in collaboration with the Academia Sinica of Taiwan.

ALMA consists of a giant array of sixty-six 12 m antennas with baselines up to 16 km, and an additional compact array of 7 m and 12 m antennas to greatly enhance ALMA's ability to image extended targets, located on the Chajnantor plateau at 5000m altitude.

ALMA is driven by three key science goals:

- Detecting spectral line emission from CO in a normal galaxy like the Milky Way at a redshift of $z = 3$, in less than 24 hours.
- Imaging the gas kinematics in protostars and in protoplanetary disks around young Sun-like stars in the nearest molecular clouds (150 pc).

- Providing precise high dynamic range images at an angular resolution of 0.1 arc-seconds

In order to accomplish those goals, ALMA will initially observe at wavelengths in the range 3 mm to 400 μm (84 to 720 GHz). The antennas can be moved around, in order to provide different baseline lengths. More extended configurations give higher spatial resolution, while more compact arrays give better sensitivity for extended sources. In addition to the array of 12 m antennas, there is the Atacama Compact Array (ACA), used to image large scale structures that are not well sampled by the ALMA 12 m array, consisting of twelve 7 m antennas and four 12 m antennas.

The current ALMA Archive design allows for a maximum raw data rate of 64 MB/s and an average data rate of 6.4 MB/s, which produces the long-term storage capacity of approximately 200 TB/year. The ALMA correlators can generate data up to 1000 MB/s. Reduced ALMA datacubes will have thousands of channels with millions of pixels, supporting multiple polarizations, and will range from tens to hundreds of gigabytes.

2.2 The Square Kilometre Array

The Square Kilometre Array[2] (SKA) is a global science and engineering project led by the SKA Organization, a not-for-profit company with its headquarters at Jodrell Bank Observatory, near Manchester. When construction starts in 2016 (according to the official schedule[3]) in Australia and in Southern Africa, thousands of linked radio wave receptors will be located to combine the signals from the antennas in each region creating a telescope with a collecting area equivalent to a dish with an area of about one square kilometer to discover how the first stars and galaxies formed after the Big Bang, how galaxies have evolved and the nature of gravity. It comprises:

- An array of dish receptors in eight African countries.

- An array of mid frequency aperture arrays in the Karoo.

[2] http://www.skatelescope.org/

[3] http://www.skatelescope.org/the-project/project/

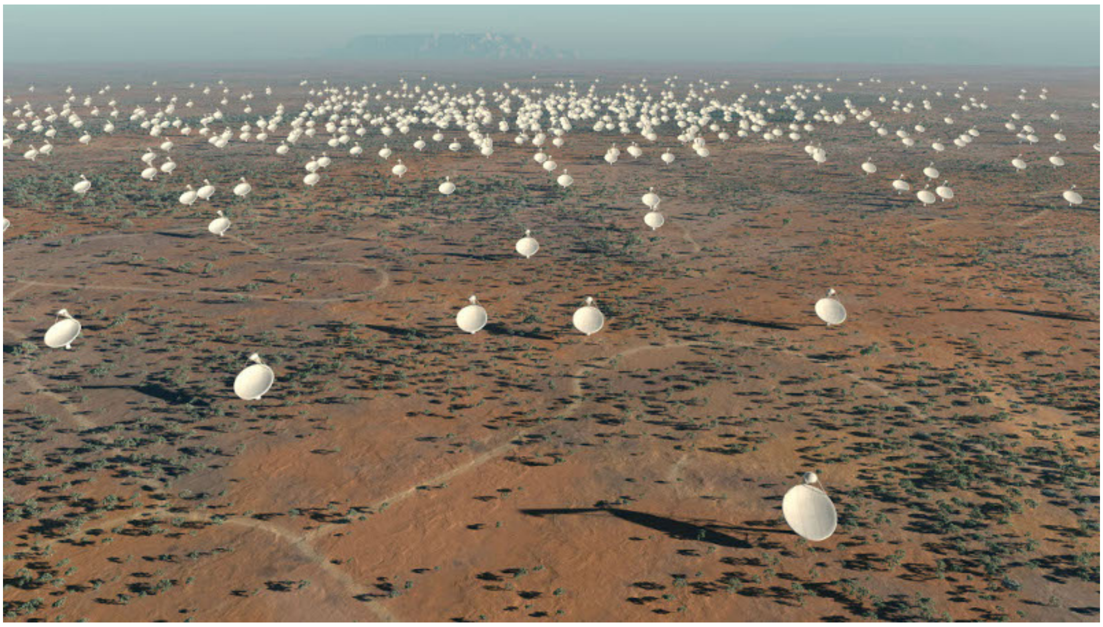


Figure 2.2: Artist's impression of the SKA dishes. Credit: SKA Organisation/TDP/DRAO/Swinburne Astronomy Productions

- A smaller array of dish receptors and an array of low frequency aperture arrays in the Murchison Radio-astronomy Observatory in Australia.

Some relevant figures and facts about the part of the Science Data Processor (SDP), the SKA system that will accept the data output from the correlator/beamformer and that will deliver final calibrated data products to scientists are listed below:

- Computing
  - SKA Phase 1 (to be procured 2017): Several hundred PetaFlop/s
  - SKA Phase 2 (2022): ExaFlop/s range
- Data rates
  - SKA input into correlator: Up to 10 - 500 TB/s
  - SKA output from correlator: 320 GB/s from dish correlator and 3 TB/s from Aperture array correlator.

This numbers make the Square Kilometre Array the *ultimate Big Data challenge* [**? ?**].

2.3 Murchison Widefield Array

The Murchison Widefield Array is one of the first Square Kilometre Array precursor to enter full operations, generating a huge amount of information that needs to be stored for later retrieval by scientists. To store the data generated by the MWA three 1 TB hard drives every two hours are needed (it will store about 3 Petabytes at the Pawsey Center each year), so the problem is not just a matter of storing the observations but how to distribute them from the MWA team in remote places, like the MIT, Victoria University of Wellington in New Zealand and India.

According to Professor Andreas Wicenec, from The University of Western Australia node of the International Centre for Radio Astronomy Research (ICRAR), the MWA now ingest data at *"more than 400 megabytes per second [. . .], streaming along the National Broadband Network from the desert 800 km away"*[4]. Data travels through a 10 gigabit per second connection between the Murchison Radio-astronomy Observatory (MRO) and Geraldton.

The data are not obviously intended to be fully available for everybody at every time: for instance, MIT researchers are interested in the early universe so filtering techniques to control what data is copied from the Pawsey Center archive to the MIT machines are used. By 2013, more than 150 TB of data had been transferred automatically to the MIT store, with a stream of up to 4 TB a day increasing that value.

2.4 Large Synoptic Survey Telescope

The Large Synoptic Survey Telescope (LSST) is a new kind of telescope whose widefield of view allows it to observe large areas of the sky at once. It can take almost 1000 panoramic images each night, covering the sky twice a week. Data from the LSST will be used to create a 3D map of the Universe with unprecedented depth and detail. Plans for sharing the data from LSST with the public are really ambitious, as it is intended that anyone with a personal computer will be able to fly through the Universe, zooming past objects a hundred million times fainter than what can be observed with the human eye.

[4] http://abcstarstuff.tumblr.com/post/56503906455

Some of the institutional members[5] are Google, Caltech, Harvard-Smithsonian Center for Astrophysics, Fermi National Accelerator Laboratory or STSI.

LSST observing will produce about 30 TB per night, leading to a total database over the ten years of operations of 60 PB for the raw data, and 30 PB for the catalog database, that will be processed using 100 TFlops. The data will be sent over existing optical fiber links from Chile to the U.S.

Currently finishing the design and development stages, it is expected to start operating in 2022.

[5] See http://lsst.org/lsst/about/members

Chapter 3

The Virtual Observatory

The International Virtual Observatory Alliance[1] (IVOA) was formed in 2002 with the aim to *"facilitate the international coordination and collaboration necessary for the development and deployment of the tools, systems and organizational structures necessary to enable the international utilization of astronomical archives as an integrated and interoperating virtual observatory."* The IVOA now comprises programs from several countries and intergovernmental organizations like ESA and ESO.

The IVOA focuses on the development of standards and encourages their implementation for the benefit of the worldwide astronomical community. As a result, a federation of lightly-coupled, interoperable web-services provide astronomers with data from multiple data archives and catalogues. Working Groups are constituted with cross-program membership in those areas where key interoperability standards and technologies have to be defined and agreed upon. The Working Groups develop standards using a process modeled after the World Wide Web Consortium, in which Working Drafts progress to Proposed Recommendations and finally to Recommendations. Recommendations may ultimately be endorsed by the Virtual Observatory Working Group of Commission 5 (Astronomical Data) of the International Astronomical Union. The IVOA also has Interest Groups that discuss experiences using VO technologies and provide feedback to the Working Groups.

In this section we are not going to make a deep study of the Virtual Observatory techniques, technologies, protocols and interfaces, just those needed and selected to explain our proposal of including NoSQL into VO. Bearing in

[1] http://www.ivoa.net/

mind that the problem we are trying to address is data modelling, we will just make an overview of those aspects related with our work.

3.1 Flexible Image Transport System

The Flexible Image Transport System (FITS) is an open standard defining a digital file format useful for storage, transmission and processing of scientific and other images. FITS is the most commonly used digital file format in astronomy. Unlike many image formats, FITS is designed specifically for scientific data and hence includes many provisions for describing photometric and spatial calibration information, together with image origin metadata. The FITS format was first standardized in 1981; it has evolved gradually since then, and the most recent version (3.0) was standardized in 2008.

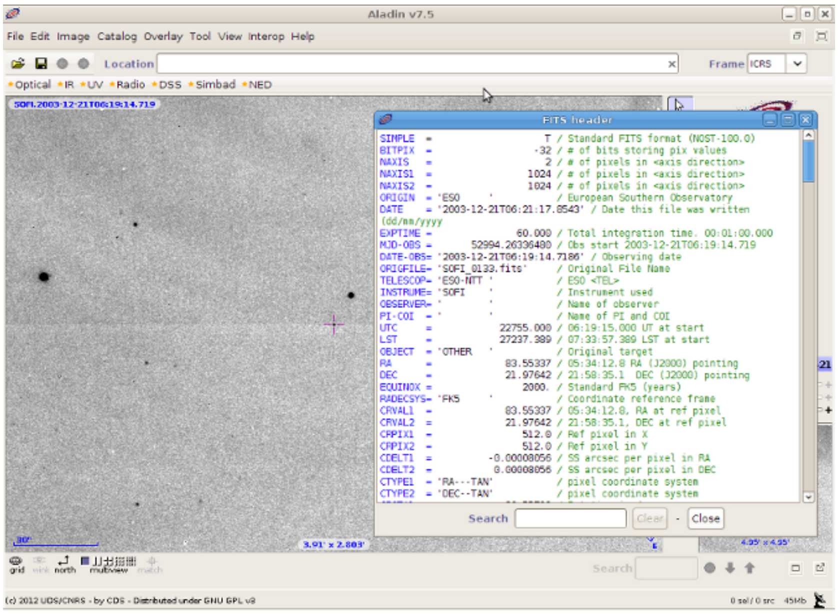
FITS is also often used to store non-image data, such as spectra, photon lists, data cubes, or even structured data such as multi-table databases. A FITS file may contain several extensions, and each of these may contain a data object. For example, it could be possible to store X-ray and infrared exposures in the same file.

FITS support is available in a variety of programming languages that are used for scientific work, including C, C++, C#, Fortran, IDL, Java, Mathematica, MatLab, Perl, PDL, or Python.

Image processing programs such as GIMP can generally read simple FITS images, but cannot usually interpret complex tables and databases.

3.1.1 FITS Data Format

Each FITS file consists of one or more headers containing ASCII card images that carry keyword/value pairs, interleaved between data blocks. The keyword/value pairs provide information such as size, origin, coordinates, binary data format, free-form comments, history of the data, and anything else the creator desires. In more technical terms, a FITS file is comprised of parts called Header Data Units (HDU), being the first HDU called primary HDU o primary array. This array can contain a 1-999 dimensional array. A typical primary array could contain a 1D spectrum, 2D image or 3D data cube. Any number of HDU can follow the main array, and are called FITS extensions. Currently,


Aladin v7.5
File Edit Image Catalog Overlay Tool View Interop Help
Location
Frame ICRS
Optical IR UV Radio DSS Simbad NED
SOFI.2003-12-21T06:19:14.719
FITS header
Search
Clear
Close
3.91' x 2.803'
4.95' x 4.95'
grid wink north multiview match
Search
(c) 2012 UDS/CNRS - by CDS - Distributed under GNU GPL v3
0 sel / 0 src 45Mb

```
SIMPLE  =                    T / Standard FITS format (NOST-100.0)
BITPIX  =                  -32 / # of bits storing pix values
NAXIS   =                    2 / # of pixels in <axis direction>
NAXIS1  =                 1024 / # of pixels in <axis direction>
NAXIS2  =                 1024 / # of pixels in <axis direction>
ORIGIN  = 'ESO     '           / European Southern Observatory
DATE    = '2003-12-21T06:21:17.8543' / Date this file was written
(dd/mm/yyyy
EXPTIME =               60.000 / Total integration time. 00:01:00.000
MJD-OBS =       52994.26336480 / Obs start 2003-12-21T06:19:14.719
DATE-OBS= '2003-12-21T06:19:14.7186' / Observing date
ORIGFILE= 'SOFI_0133.fits'     / Original File Name
TELESCOP= 'ESO-NTT '           / ESO <TEL>
INSTRUME= 'SOFI    '           / Instrument used
OBSERVER= '        '           / Name of observer
PI-COI  = '        '           / Name of PI and COI
UTC     =            22755.000 / 06:19:15.000 UT at start
LST     =            27237.389 / 07:33:57.389 LST at start
OBJECT  = 'OTHER   '           / Original target
RA      =             83.55337 / 05:34:12.8 RA (J2000) pointing
DEC     =             21.97642 / 21:58:35.1  DEC (J2000) pointing
EQUINOX =                2000. / Standard FK5 (years)
RADECSYS= 'FK5     '           / Coordinate reference frame
CRVAL1  =             83.55337 / 05:34:12.8, RA at ref pixel
CRVAL2  =             21.97642 / 21:58:35.1, DEC at ref pixel
CRPIX1  =                512.0 / Ref pixel in X
CRPIX2  =                512.0 / Ref pixel in Y
CDELT1  =          -0.00008056 / SS arcsec per pixel in RA
CDELT2  =           0.00008056 / SS arcsec per pixel in DEC
CTYPE1  = 'RA---TAN'           / pixel coordinate system
CTYPE2  = 'DEC--TAN'           / pixel coordinate system
```

Figure 3.1: Viewing a FITS header in Aladin

three different extensions can be defined:

- Image extension, a 0-9999 dimensional array of pixels, which begins with `XTENSION = 'IMAGE'`
- ASCII table extension which stores tabular data in ASCII formats. They begin with `XTENSION = 'TABLE'`
- Binary table extension stores tabular data in binary representation. Headers start with `XTENSION = 'BINTABLE'`

Besides, there are additional type of HDU called random groups, but only used for radio interferometry.

3.2 Table Access Protocol

The Table Access Protocol [**?**] (TAP) defines a web service for accessing tables containing astronomical catalogues. TAP is the protocol which underlies in the process of posing a query against a data source (or several data sources). The result of a query is a table, usually a VOTable.[2]

Queries which use TAP protocol can be made through several clients, like:

[2] Support for VOTable output is mandatory, while other formats may be available.

- TOPCAT[3]

- TAPHandle[4] is a TAP client which operates fully within the Web browser

- The TAP shell[5], a command line interface to querying TAP servers, complete with metadata management and command line completion.

- The GAVO VOTable library[6], which allows embedding TAP queries in Python.

The types of queries implemented by TAP are:

- data queries, where the query result is directly the science data to be operated upon;

- metadata queries, where the query result is metadata of the datasets, the archival service, or both; and

- Virtual Observatory Support Interface (VOSI) queries, which provide information on the different TAP service capabilities.

TAP includes support for multiple query languages, including queries specified using the Astronomical Data Query Language (ADQL; see [?]) and the Parameterised Query Language [?] (PQL, under development). Other query languages are also supported, and this mechanism allows developments outside the IVOA to be used without modifying the TAP specification. Finally, it also includes support for both synchronous and asynchronous queries. Special support is provided for spatially indexed queries using the spatial extensions in ADQL.

Listing 3.1 shows an ADQL sample query: a crossmatch between a user-uploaded catalog (`TAP_UPLOAD.T1`), and the Third US Naval Observatory CCD Astrograph Catalog[7] (UCAC3) catalog, compensating the fit for proper motion.

[3] http://www.star.bris.ac.uk/~mbt/topcat/

[4] http://saada.unistra.fr/taphandle/

[5] http://vo.ari.uni-heidelberg.de/soft/tapsh

[6] http://docs.g-vo.org/DaCHS/votable.html

[7] http://www.usno.navy.mil/USNO/astrometry/optical-IR-prod/ucac

Listing 3.1: ADQL sample query: the results will be a crossmatch between a user-uploaded catalog, and the UCAC3 catalog, compensating the fit for proper motion.

```
SELECT
 u.raj2000+d_alpha+d_pmalpha/cos(radians(u.dej2000))*(u.epoch-2000) AS
    ra_icrs,
 u.dej2000+d_delta+d_pmdelta*(u.epoch-2000) AS de_cicrs,
 u.pmra+d_pmalpha AS pmra_icrs,
 u.pmde+d_pmdelta AS pmde_icrs,
 u.*
FROM
  TAP_UPLOAD.T1 AS u
  JOIN ucac3.icrscorr AS c
  ON (c.alpha=FLOOR(u.raj2000)+0.5 and c.delta=FLOOR(u.dej2000)+0.5)
```

3.2.1 ObsCore and ObsTAP

A data model is a description of the objects represented by a computer system with their properties and relationships, a logical model detailing the decomposition of a complex dataset into simpler elements, like a collection of concepts and rules used in defining data model.

In 2011 IVOA proposed a new recommendation: Observation Data Model Core Components [?] (ObsCore). ObsCore is a generic data model for the metadata necessary to describe any astronomical observation. Practically all of the ObsCore metadata (which is modelled as a single table with multiple columns) can be found in the FITS headers of reduced datasets.

The ObsCore document was intended not just to be a description of said data model, but as an interface specification which integrated the data modeling and data access aspects in a single service, called ObsTAP. In other words, ObsTAP is the combination of the TAP with the ObsCore data model.

3.2.2 Universal Worker Service

The Universal Worker Service [?] (UWS) standard defines a work pattern that specifies how to build asynchronous (as the client does not wait for each request to be fulfilled; even if the client disconnects from the service, activity is not aborted), stateful (the service remembers results of a previous activity),

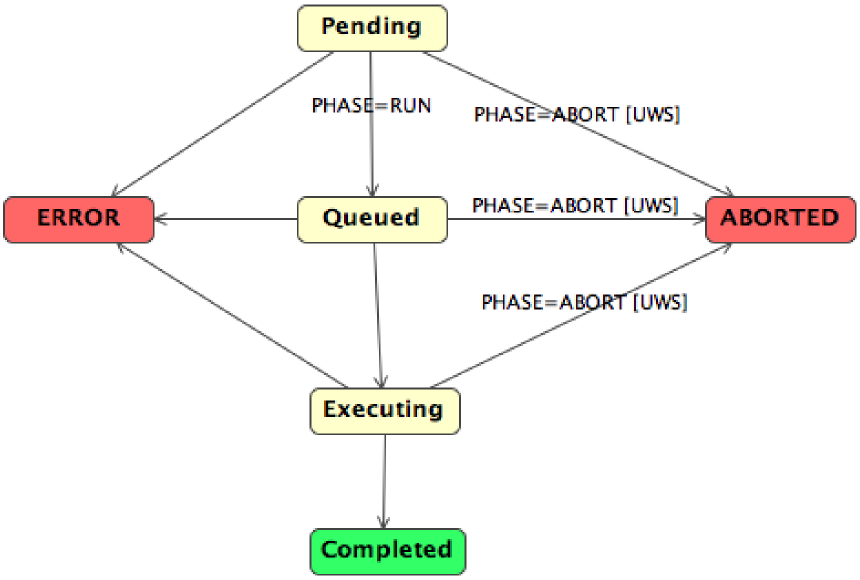


Figure 3.2: Universal Worker Server Job states

job-oriented services[8].

When the execution starts, the job can adopt several states, following the state machine illustrated in Fig. 3.2.

3.3 OpenCADC

The OpenCADC[9] is a software framework, developed by the Canadian Astronomy Data Center (CADC), and which implements a number of VO standards. It can be used either directly or as a library to implement VO services.

OpenCADC is used for implementing the query capabilities of the ASA (see Fig. 4.4). It is composed of several libraries, but the most important from the point of view of their interface are cadcUWS, which implements the UWS pattern, and cadcTAP, which implements the TAP web interface.

3.3.1 cadcUWS library

We will briefly describe the structure of the code relating UWS in OpenCADC (the cadcUWS Library), the section of the code provides Job class and plugin

[8] Job specification rules following the UWS pattern are written in the Job Description Language (JDL).

[9] https://code.google.com/p/opencadc/source/browse

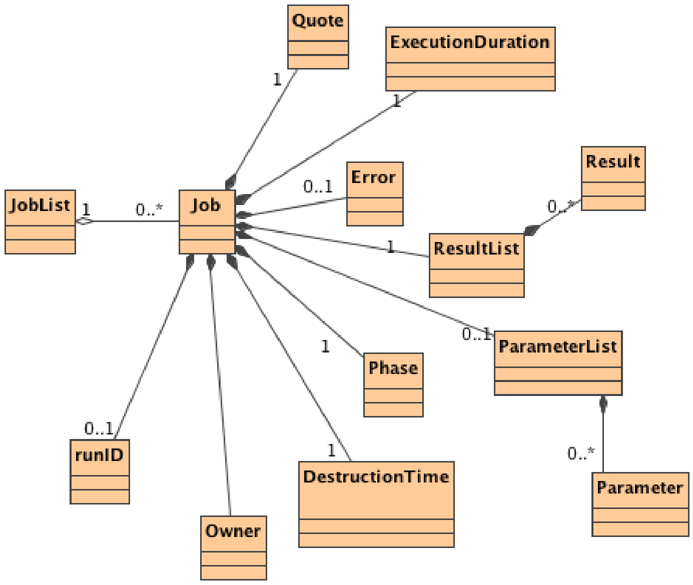


Figure 3.3: Class diagram representing UWS objects

architecture, servlet with UWS async and sync behavior. These classes are illustrated in Fig. 3.3, and listed below.

JobManager Class responsible for job control.

JobPersistence Class in In charge of storing and retrieving Job state.

JobExecutor Class which executes every job in separated threads.

JobRunner The code that actually executes the job.

3.3.2 cadcTAP library

The cadcTAP library is responsible of async and sync queries, where QueryRunner implements JobRunner. It also contains TAP_SCHEMA[10] Data Definition Language[11] (DDL) statements and is used by query parser to validate table and column usage.

TapQuery Interface It has a separate implementation for each LANG (e.g. LANG=ADQL) specified and processes the query to local SQL.

[10] TAP services try to be self-describing about what data they contain. They provide information on what tables they contain in special tables in TAP_SCHEMA

[11] In RDBMS, the Data Definition Language is the language that allows for the creation of tables, fields, and data types.

SqlQuery When code states `LANG=SQL`, it implements TapQuery and fully navigates it (`FROM`, `WHERE` and `HAVING` clauses).

AdqlQuery Same as before when `LANG=ADQL`.

Plugins The QueryRunner class (below) needs to be able to find the following classes:

- **UploadManager** Interface for classes responsible for uploading local data to the remote TAP service.
- **TableWriter** Interface for classes that convert the retrieved data into a particular data format.
- **FileStore** Interface for classes that deal with the local file storage.

QueryRunner Class that implements JobRunner and sets Job state, finds a DataSource and uses TapSchema, UploadManager, TapQuery and TableWriter.

Chapter 4

Integrating NoSQL in the Virtual Observatory

Modern relational databases have shown little efficiency in certain applications using intensive data, like indexing of a large number of documents, sites rendering with high traffic, and streaming sites.

Typical RDBMS implementations are tuned either for small but frequent reads and writes or a large set of transactions that have few write accesses. On the other hand, NoSQL DB are optimized for read and bulk-ingest operations, and perform outstandingly where a relational DB would slow down. They are especially fast when large amounts of data have to be queried, and if we consider that speed is more important than consistency[1].

As mentioned in the chapter on the Virtual Observatory, the IVOA has a standard called Astronomical Data Query Language (ADQL) that users do not necessarily need to know, because they can pose a query with a GUI, as long as the query is translated into standard ADQL. The receiving service likewise converts the standard ADQL into whatever its own database servers need, but always inside the relational model. For the reasons stated in Chapter 2, we propose the use of a different approach, the NoSQL technology.

[1] Consistency is used here in the sense of having changes replicated to multiple nodes at the same time. RDBMS ensure that consistency, but that makes them less performant. Given that VO users do not know beforehand whether some datasets are available or not, speed in data retrieval and operations is more important than consistency (which will not be perceived unless for special circumstances).

4.1 Introduction to NoSQL

A non-relational database just stores data without explicit and structured mechanisms to link data from different buckets to one another.

NoSQL implementations used in the real world include 3TB Digg social media website[2] green markers (indicated to highlight the stories voted by others in the social network), the 6 TB of the ENSEMBLES[3] European Commission Joint Research Centre (JRC) database used in comparing models and air quality, and the 50 TB of Facebook inbox search.

NoSQL architectures often provide limited consistency, such as events or transactional consistency restricted to only data items. Some systems, however, provide all guarantees offered by systems complying with the Atomicity, Consistency, Isolation and Durability (ACID) criteria by adding an intermediate layer. The ACID properties guarantee the reliable processing of database transactions, but they are not needed for system where the data are not subject to transactions, such as read-only systems.

There are two systems that have been deployed and provide for storage of snapshot isolation column: Google Percolator (based on BigTable system) and Hbase transactional system developed by the University of Waterloo. These systems use similar concepts in order to achieve distributed multiple rows ACID transactions with snapshot isolation guarantees for the underlying storage system in that column, with no extra overhead in data management, no system deployment middleware or any maintenance introduced by middleware layer.

Many NoSQL systems employ a distributed architecture, maintaining data redundantly on multiple servers, often using distributed hash tables. Thus, the system may actually escale adding more servers, and thus a server failure may be tolerated.

There are different NoSQL DBs for different projects:

Document oriented These are systems whose main purpose is to record document data and metadata, such as text corpuses, massive mail systems, or similar datasets.

[2] http://digg.com

[3] http://ensemble2.jrc.ec.europa.eu/public/

- CouchDB[4]
- MongoDB[5]
- RavenDB[6]
- IBM Lotus Domino[7]

Graph oriented These are systems whose emphasis is made in recording the relationships (including directionality) between the objects in the data store.

- Neo4j[8]
- AllegroGraph[9]
- InfiniteGraph[10]
- Sones GraphDB[11]
- HyperGraphDB[12]

Key-value oriented These are systems where simple key-value pairs (such as those in FITS headers) are stored, withing a shallow hierarchy of metadata.

- Cassandra[13]
- BigTable[14]
- Dynamo (Amazon)[15]
- MongoDB[16]
- Project Voldemort (LinkedIn)[17]

[4] http://couchdb.apache.org
[5] http://www.mongodb.org
[6] http://ravendb.net
[7] http://www.ibm.com/software/products/us/en/ibmdomino/
[8] http://www.neo4j.org
[9] http://www.franz.com/agraph/allegrograph/
[10] http://www.objectivity.com/infinitegraph
[11] https://github.com/sones/sones
[12] http://www.hypergraphdb.org/
[13] http://cassandra.apache.org/
[14] http://research.google.com/archive/bigtable.html
[15] http://aws.amazon.com/dynamodb/
[16] http://www.mongodb.org/
[17] http://www.project-voldemort.com/voldemort/

Multivalue

- OpenQM[18]

Object Oriented These systems try to store complex objects, their metadata and relationships in ways that are directly usable by software systems to save and retrieve internal state.

- Zope Object Database [19]
- db4o[20]
- GemStone S[21]
- Objectivity/DB[22]

Tabular These systems provide support for storing very large tables, but not for relationships between them.

- HBase[23]
- BigTable
- LevelDB (BigTable open version)[24]
- Hypertable[25]

All of these systems can run on clusters of inexpensive machines, or can be run on distributed cloud systems such as the Amazon Elastic Cloud, Google App Engine, or similar.

4.2 Advantages and uncertainties of NoSQL

Elastic scaling DBA have relied on scale up buying bigger servers as database load increases rather than scale out distributing the database across

[18] http://www.openqm.com/cgi/lbscgi.exe?t0=h&t1=home
[19] http://www.zodb.org/en/latest/
[20] http://www.db4o.com/
[21] http://gemtalksystems.com/index.php/products/gemstones/
[22] http://www.objectivity.com/
[23] http://hbase.apache.org/
[24] http://code.google.com/p/leveldb/
[25] http://hypertable.org/

multiple hosts as load increases. However, as transaction rates and availability requirements increase, and as databases move into the cloud or onto virtualized environments, the economic advantages of scaling out on commodity hardware become irresistible.

Unlike RDBMS, the NoSQL databases are designed to expand transparently to scale and they are usually designed with low-cost in mind.

Economics NoSQL databases typically use cheap clusters servers, while RDBMS tends to rely on expensive proprietary servers and storage systems. The result is a lower cost per transaction/second for NoSQL.

Flexible data models Even minor changes to the data model of an RDBMS have to be carefully managed and may necessitate downtime or reduced service levels. NoSQL databases have a less strict (in the worst case) data model restrictions. NoSQL allows the application to store virtually any structure it wants in a data element. The result is that application changes and database schema changes do not have to be managed as one unit. In theory, this will allow applications and user facing services to iterate faster.

No need for DBAs RDBMS systems can be maintained only with the assistance of expensive and highly trained DBAs, while NoSQL databases are generally designed to require less management: automatic repair, data distribution, and simpler data models lead to lower administration and tuning requirements and costs.

NoSQL systems have generated a lot of enthusiasm, but there are still a lot of questions about its future:

Maturity RDBMS systems have been around for a long time. NoSQL evangelists will argue that their advancing age is a sign of their obsolescence, but mature RDBMS systems are stable. Most NoSQL alternatives are in pre-production versions with many key features yet to be implemented.

Support Most NoSQL systems are open source projects, and although there are usually one or more firms offering support for each NoSQL database, these companies often are small start-ups. There are risks in choosing a NoSQL solution that is no longer supported in the future, but at the same time, if the community is large enough, NoSQL systems can maintain their deployability for longer time.

Expertise There are millions of developers with expertise using RDBMS. On the other hand, NoSQL developers are, by far, a minority. This situation seems to be changing, but for now, it is easier to find experienced RDBMS programmers or DBAs than NoSQL experts.

4.3 NoSQL deployments in scientific research

To support our proposal of implementing NoSQL systems in the VO, we present in this subsection some successful case studies which show that selecting such a new alternative (in comparison with the classical RDBMS) for big scientific projects is not necessarily a risk, the main reason argued against NoSQL.

4.3.1 CMS at the LHC: MongoDB

High-energy physicists working at the Compact Muon Solenoid (CMS) detector at the LHC, that generates more than 100 datacentres in a three-tier model and generates around 10PB of data each year, are benefiting from a NoSQL database management system that gives them unified access to data from a huge variety of sources (from relational and non-relational data sources, such as relational databases, document-oriented databases, blogs, wikis, file systems and customised applications). The team providing data management to the CMS Cern project has built a system using MongoDB in preference to relational database technologies and other non-relational options. The reasons to select MongoDB were:

- Dynamic queries support
- Full indexes
- Auto-sharding

Several years ago the data management group at the CMS confronted a data discovery problem, with a variety of databases necessitating a user interface that would hide the complexity of the underlying architecture from the physicists. There was a vast variety of distributed databases and different formats (HTML, XML, JSON files, etc.).

To provide the ability to search and aggregate information across this complex data landscape CMS's Data Management and Workflow Management (DMWM) project created a Data Aggregation System (DAS), built on top of MongoDB. According to Valentin Kuznetsov, a research associate at Cornell University, and team member, *"there was nothing specific to the system related to our experiment"*[26], so it can be deduced that this MongoDB approach is extensible.

4.3.2 ATLAS Workload Management System: Cassandra

A Thoroidal LHC Apparatus (ATLAS) is another of the six particle detectors experiments at LHC whose data handling is a real challenge: almost 1 billion proton-proton interactions per second are recorded, leading to more than 10PBytes per year (it actually generates 1PByte of raw data each second, before filtering). By 2014, the expected data rate will be 40 PBytes per year.

The reason to select a NoSQL solution is that there is no need to store the monitoring data in a RDBMS, deciding instead a system which can provide a very high degrees of availability, resilience and scalability.

At first stage, two of the most popular platforms were considered: Apache Cassandra[27] and Apache Hbase[28].

Cassandra was finally selected, as its learning curve is less steep and for ease of operation,[29] and considering that there was already a Cassandra deployment within the ATLAS collaboration, the solution was taken.

As a matter of fact, in the tests performed, Cassandra's time per extracted entry was 10 ms, with 10 concurrent clients. This means 1000 queries per second, about twice the rate currently experienced by their Oracle DB. An analogous test done against Oracle (from a Python client) yielded roughly 100ms per query[30].

[26] http://www.computerweekly.com/news/2240166469/Cornell-Cern-project-plumps-for-NoSQL-DBMS

[27] http://cassandra.apache.org/

[28] http://hbase.apache.org

[29] http://cds.cern.ch/record/1446655/files/ATL-SOFT-PROC-2012-012.pdf

[30] See http://cds.cern.ch/record/1446655/files/ATL-SOFT-PROC-2012-012.pdf

4.3.3 Measuring radiation levels in Seattle: CouchDB

Researchers at the University of Washington used the Cloudant [31] database as part of an experiment that determined radiation levels in Seattle as a result of the recent Fukushima nuclear disaster are *"well below alarming limits."*[32] The research team, which includes Cloudant Founder and Chief Scientist Mike Miller studied particles captured from the five air filters and used Cloudants CouchDB-based BigCouch database to store and process the data.

They chose Cloudant because with it, it was possible to start monitoring radiation levels very quickly, after a very easy setup and the capability of sampling gigantic quantities of air, allowing them to analyze the data and sharing it in near-real-time, benefiting from BigCouch's MapReduce engine.

4.4 Choosing a NoSQL database: MongoDB

MongoDB (from *humongous*) is an open source document-oriented NoSQL database system developed and supported by 10gen. Instead of storing data in tables as is done in a *classical* relational database, MongoDB stores structured data as JSON-like documents with dynamic schemas (MongoDB calls the format BSON), making the integration of data in certain types of applications easier and faster.

An example of Mongo DB document insertion is shown in Listing 4.1.

10gen began Development of MongoDB in October 2007 and was not created to be just another database that tries to do everything for everyone. Instead, MongoDB was created to work with documents rather than rows, was extremely fast, massively scalable, and easy to use. In order to accomplish this, some RDBMS features were excluded, namely the support for transactions.

A document database is more like a collection of documents. Each entry is a document, and each one can have its own structure. If you want to add a field to an entry, you can do so without affecting any other entry.

4.4.1 Justification

We have decided to use MongoDB for these reasons:

[31] https://cloudant.com/

[32] http://gigaom.com/2011/03/31/how-nosql-is-helping-allay-seattles-radiation-fears/

Listing 4.1: MongoDB column insertion example in JSON

```
db.columns.insert( {
              table_name: "TAP_SCHEMA.schemas",
              column_name: "schema_name",
              utype: null,
              ucd: null,
              unit: null,
              description: "schema name for reference to
                  TAP_SCHEMA.schemas",
              datatype: "VARCHAR",
              size: 64,
              principal: 1,
              indexed: 0,
              std: 0
       }
);
```

- It is open source code (available at https://github.com/mongodb/mongo).
- It is widely used in the NoSQL community.
- It has full index support, which is a must for large datasets
- It is very easy to install and to connect through drivers with several programming languages like Java, Python, Ruby, C/C++, PHP or Scala.
- It supports the MapReduce paradigm.
- It is possible to connect MongoDB with Hadoop (the *de facto* big data processing and analytic platform [33], allowing to to send MongoDB data into Hadoop MapReduce jobs, process the data and return it back to a MongoDB collection).

The following subsection provides a more detailed list of MongoDB features.

4.4.2 Main features

Ad-hoc queries MongoDB supports search by field, range queries, regular expression searches. Queries can return specific fields of documents and

[33] http://hadoop.apache.org

also include user-defined JavaScript functions.

Indexing Any field in a MongoDB document can be indexed (indices in MongoDB are conceptually similar to those in RDBMS).

Data replication MongoDB supports master-slave replication. A master can perform reads and writes. A slave copies data from the master and can only be used for reads or backup (not writes). The slaves have the ability to select a new master if the current one goes down.

Load balancing MongoDB scales horizontally using sharding (a shard is a master with one or more slaves). The developer chooses a shard key, which determines how the data in a collection will be distributed[34]. The data is split into ranges (based on the shard key) and distributed across multiple shards. MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure. Automatic configuration is easy to deploy and new machines can be added to a running database.

File storage capabilities MongoDB could be used as a file system, taking advantage of load balancing and data replication features over multiple machines for storing files. This feature, GridFS, is included with MongoDB drivers and available for development languages. GridFS is used, for instance, in plugins for NGINX and lighttpd. In a multi-machine MongoDB system, files can be distributed and copied multiple times between machines transparently, thus effectively creating a load balanced and fault-tolerant system.

Aggregation MapReduce can be used for batch processing of data and aggregation operations. The aggregation framework enables users to obtain the same results as the SQL `GROUP BY` clause.

Once we have seen the main features of MongoDB, we can move on to the system itself [**? ? ?**].

[34] For instance, astronomical collections can be naturally split in different sky stripes, object types, magnitude ranges, et cetera.

Relational	Document oriented
Table	Collection
Row	Document
Column	Field

Table 4.1: Correspondence between relational and NoSQL terminology

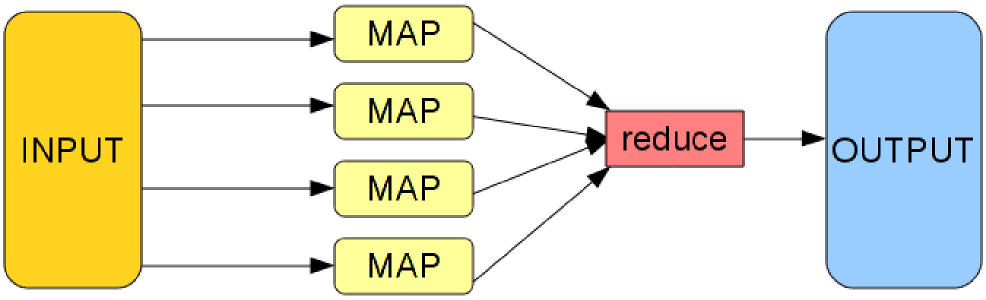


Figure 4.1: MapReduce concept

4.4.3 MongoDB concepts

We must know four concepts to understand MongoDB: Database, Collection, Document, and Field. The correspondence between the document-oriented MongoDB terminology, and that of relational systems is shown in Table 4.1

SQL/Mongo syntax differences

In this subsection we can see some differences in the DDL syntax used by both SQL and MongoDB. For the latter, we can call it from any language having a driver to connect to the DB or use the interactive shell (the code presented here uses this shell). Listing 4.2 shows the differences in syntax between the DDL instructions for SQL and MongoDB.

Listing 4.2: Syntax comparison between SQL and MongoDB BSON for table/-document creation, document/row insertion, document/row selection, document/row deletion, and document/row updates.

```
-- CREATE and INSERT
CREATE TABLE tap_schema.schemas (
       schema_name varchar(64),
       utype       varchar(512) NULL,
       description varchar(512) NULL,
       primary key (schema_name)
);

INSERT INTO tap_schema.schemas (schema_name,description,utype) VALUES
( 'TAP_SCHEMA', 'a special schema to describe a TAP tableset', null );

db.schemas.insert( {
       schema_name: "TAP_SCHEMA",
       utype: null,
       description: "a special schema to describe a TAP tableset"}
);

-- SELECT
SELECT *
  FROM tap_schema.schemas t
 WHERE schemas.schema_name = 'TAP_SCHEMA';

db.tables.find({schema_name: "TAP_SCHEMA"});

-- DELETE

DELETE FROM TAP_SCHEMA.SCHEMAS WHERE schemas.schema_name =
    'TAP_SCHEMA';

db.tables.remove({schema_name: "TAP_SCHEMA"});

-- UPDATE

UPDATE TAP_SCHEMA.SCHEMAS
   SET schemas.description = 'desc'
 WHERE schemas.schema_name = 'TAP_SCHEMA';

db.tables.update(
       {schema_name = 'TAP_SCHEMA'},
       {$set: {description = "desc"}},
       {multi:true}
);
```

4.4.4 MongoDB MapReduce support

MapReduce, developed by Google, is a programming model and its implementation for processing (*e.g.* raw data from telescopes) and producing (*e. g.* representations of graph structure of systems) huge amounts of data. As the runtime controls the partitioning of the input data, controlling the program execution across several hundred or thousands of nodes and even controlling machine failures, the developers with low or none expertise at all in parallel programming can take advantage of this model with a very low learning curve. The concept is illustrated in Fig. 4.1.

MapReduce is usually used to solve problems involving big size datasets, up to several petabytes. For this reason, this model is used in distributed file systems, like the Hadoop Distributed File System (HDFS)[35].

The MapReduce model allows to write a *map* function which takes a key/-value pair, operates on it (for instance, performs FITS-file multi-header and/or multi-table calculations) and returns a new set of key/value pairs (source list of FITS files with values above a given threshold). The *reduce* function then aggregates/merges all the intermediate data (builds an object catalog removing duplicates from the source list, for instance).

Problems in different domains can be parallelized via MapReduce, by a suitable definition of both the *map* and *reduce* functions. In particular, tasks whose only changes are the input values can be *map*ped to the input arguments and output arguments, and the *reduce* function can be a trivial summation of null outputs from the *map* functions, while the important result has been saved by the *map*ped function (in a save-file, for instance).

Listing 4.3 shows how to define a MapReduce job in MongoDB.

4.5 Creating a FITS store in MongoDB

The first issue when working with FITS files is its inadequacy for storing information. In XXI century is difficult to maintain such storage model. So the first and more obvious solution would be replacing FITS headers with their counterparts in a non relational database.

Another issue when working with FITS format is the great amount of pos-

[35] http://hadoop.apache.org/docs/stable/hdfs_design.html

Listing 4.3: MapReduce job example

```
// First, we define the map function, registering the full date,
// the server name and number of times the user logged into the system
    (more
//  than 100, more than 1000 times

map = function Map() {
       var day = new Date(this.datetime.getYear(),
                                 this.datetime.getMonth(),
                                 this.datetime.getDate(),
                                 this.datetime.getHours());

       emit({day: day, server: this.server},
               {count:1, time: this.time});
}

// Second, we define the reduce function

reduce = function Reduce(key, arr_values) {
       var result = {count: 0, countmore100: 0, countmore1000: 0};

       for(var i in values) {
               if(values[i].timetaken > 10000) {
                       result.countmore100 += values[i].count;
               } else if(values[i].timetaken > 100000) {
                       result.countmore1000 += values[i].count;
               }
       }
       return result;
}

// Finally, we send these two functions to the server with the desired
// filter, specifying where the data must be sent

Loggin.collection.map_reduce(map, reduce,
       {
               :query => {
                       :datetime => {'$gte' => datefilterfrom},
                       :datetime => {'$lte' => datefilterto}
               },

               :out => { reduce: "queriesbyday"}
       }
)
```

sible key-value pairs in FITS headers. A possible solution for this problem could be use the features of MongoDB to translate FITS keywords into collections. So, we could create as many documents as needed, and storing them into MongoDB. Should we need to create supertypes of FITS headers, relations are also available in MongoDB through document linking (with no physical restriction in the sense of relational constraint). In this situation the schema flexibility means that we can model documents that share a similar structure but not enforced to have to same.

Another related problem is having multiple FITS formats[36], representing the users need for storing and handling the different information requirements:

- FITS Interferometry Data Interchange (FITS-IDI) Convention: conventions adopted for registering information from interferometric telescopes recordings. Used by the VLBA.
- Single Dish FITS (SDFITS) Convention for Radio Astronomy Data: conventions used for the interchange of single dish data in radio astronomy.
- Multi-Beam FITS Raw Data Format Convention: conventions used at the IRAM 30m and APEX 12m mm/submm telescopes.
- OIFITS: A Data Standard for Optical Interferometry Data: conventions used for exchanging calibrated, time-averaged data from astronomical optical interferometers.

We show now a logical layout using MongoDB features to solve the previously posed problem. Let's suppose we have several FITS files and we have to handle them as easy as possible.

For the sake of simplicity and clarity, we have not included all FITS keywords, focusing on the abstraction.

Several proposals for storing FITS header metadata in a database have been made. One of the examples is the ESO Keyword Repository [**?**], a metadata database containing all information stored in almost 10 million FITS file headers using Sybase IQ server[37], a columnar storage RDBMS.

[36] The IAU FITS Working Group (IAU-FWG) is the international control authority for the FITS (Flexible Image Transport System) data format: `http://fits.gsfc.nasa.gov/iaufwg/iaufwg.html`

[37] `http://www.sybase.com/products/datawarehousing/sybaseiq`

Listing 4.4: FITS sample.

```
SIMPLE = T / Standard FITS format
BITPIX = 8 /
NAXIS = 0 /
EXTEND = T /
BLOCKED = T /
OBJECT = 'BINARYTB' /
TELESCOP= 'VLBA ' /
CORELAT = 'VLBA ' / added to header dump by EWG
FXCORVER= '4.22 ' /
OBSERVER= 'BL146 ' /
ORIGIN = 'VLBA Correlator' /
DATE-OBS= '2007-08-23' /
DATE-MAP= '2007-08-31' / Correlation date
GROUPS = T /
GCOUNT = 0 /
PCOUNT = 0 /
END
```

For this work, we can profit from MongoDB document-oriented storage support (collections, documents and fields). MongoDB allows to define as many kinds of documents as needed, sort of meta-documents, considering that there is no compulsory for them having the same structure.

Solution Using just one collection and n documents, one for each kind. If we are going to work with FITS-IDI, SDFITS, MBFITS and OIFITS, we can use code like to create different variations of what a FITS document is like.

We start from a very simple FITS header like:

Prior to translating it into MongoDB, we create several collections in Listing 4.5, one for each different FITS types.

Now, inserting the primary HDU for a FITS-IDI header in MongoDB could not be easier, given the fact that FITS headers are collections of key-value pairs.

This code is very easy to translate into MongoDB syntax, as shown in Listing 4.6.

Note that we have specified a field named *supertype*, which acts as a foreign key in a RDMS. This field help to classify FITS files, despite MongoDB collections do not enforce the structure of its documents.

Now, we have our FITS header into a database with a document structure.

Listing 4.5: MongoDB BSON code for creating the different FITS document types.

```
db.fits_type.insert(
  {convention: "FITS-IDI"}
);

db.fits_type.insert(
  {convention: "SDFITS"}
);

db.fits_type.insert(
  {convention: "MBFITS"}
);

db.fits_type.insert(
  {convention: "OIFITS"}
);
```

Listing 4.6: MongoDB BSON code for adding FITS metadata to the NoSQL database.

```
db.fits_docs.insert(
    {convention: "FITS-IDI",
    SIMPLE: "T",
    BITPIX: 8,
    NAXIS: 0,
    EXTEND: "T",
    BLOCKED: "T",
    OBJECT: "BINARYTB",
    TELESCOP: "VLBA ",
    CORELAT: "VLBA ",
    FXCORVER: "4.22 ",
    OBSERVER: "BL146 ",
    ORIGIN: "VLBA Correlator",
    DATE_OBS: "2007-08-23",
    DATE_MAP: "2007-08-31",
    GROUPS: "T",
    GCOUNT: 0,
    PCOUNT: 0}
);
```

We keep the same approach but in a very more usable and efficient way.

4.6 Scaling the ALMA Science Archive: Bridging OpenCADC and NoSQL

As stated in [**?**], the purpose of the ALMA archive is to provide services for:

- Persistent archiving and retrieval for observational data.
- Search and retrieval of observations via several descriptors.
- Retrieval of reduced datacubes produced by the pipeline.
- Search and retrieval of technical and environmental data.

The main objective of the ALMA Archive conceptual design (see Fig. 4.2) is to guarantee that three ALMA Regional Centres (North America, East Asia and Europe) hold an identical copy of the archive at the Joint ALMA Observatory in Santiago.

The ALMA Front-end Archive (AFA; the part of the archive directly accessed by the ALMA correlator) is optimized for storage and preservation, not for data query and retrieval.

The ALMA Science Archive (ASA), on the other hand, is the user-facing part of the ALMA Archive, the interface for querying ALMA data. The ASA receives a transformed copy of the data from the AFA, because only a subset of the data and metadata in the AFA is considered to be queriable by the users.

Despite all ALMA information can be accessed from the Archive, ASA provide a optimized way to access the data in a scientific way. One of the requirements of ASA design was to provide a search and query tool which allows several parameters (position, frequency, telescope related parameters, etc.). It has two major components: the DB, which is a plain relational database with denormalized structure and the interface (shown in Fig. 4.3), built as a web application.

The VO Technologies used in the ALMA Archive are (discussed in Chapter 3):

1. VO Data Models

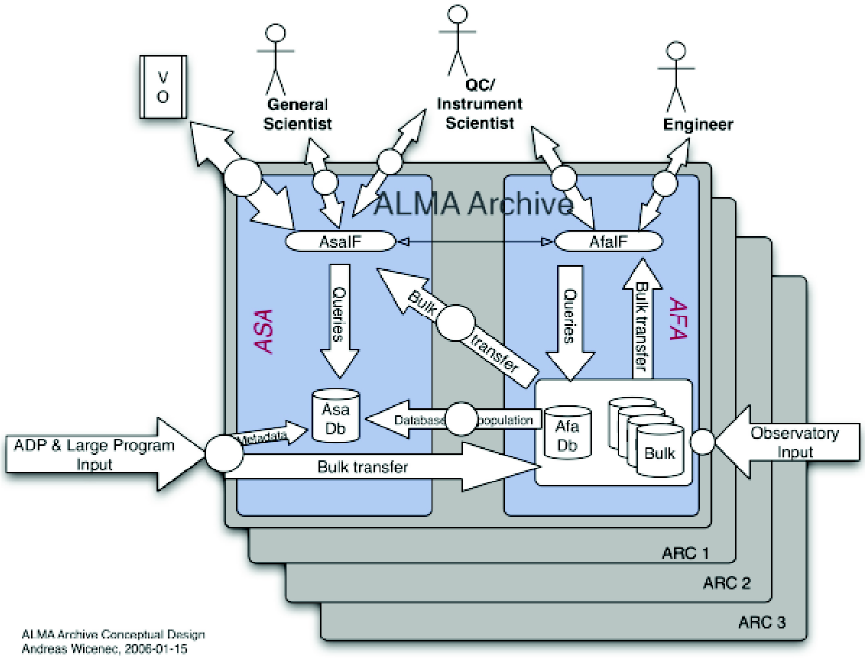
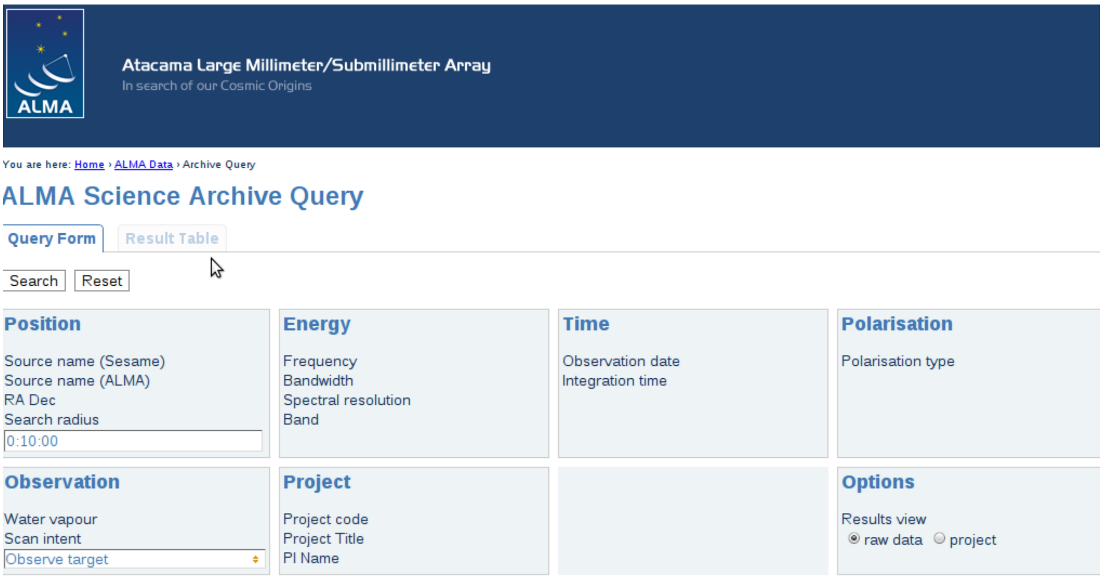


Figure 4.2: ALMA Science Archive and ALMA Front-end Archive; figure by A. Wicenec.



Atacama Large Millimeter/Submillimeter Array
In search of our Cosmic Origins

You are here: Home › ALMA Data › Archive Query

ALMA Science Archive Query

Query Form | Result Table

Search | Reset

Position	**Energy**	**Time**	**Polarisation**
Source name (Sesame) Source name (ALMA) RA Dec Search radius 0:10:00	Frequency Bandwidth Spectral resolution Band	Observation date Integration time	Polarisation type
Observation	**Project**		**Options**
Water vapour Scan intent Observe target	Project code Project Title PI Name		Results view ● raw data ○ project

Figure 4.3: ALMA Science Archive Query Interface

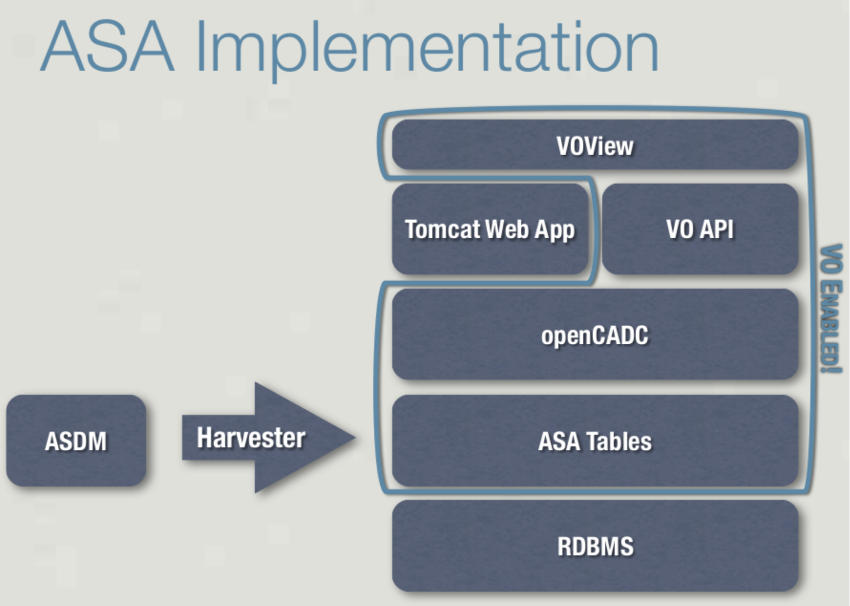


Figure 4.4: ALMA implementation diagram, courtesy of Juande Santander

2. Software libraries
   (a) OpenCADC, the framework described in Section 3.3, to provide support for TAP, UWS, and other VO technologies.
   (b) VOView[38]: a JQuery-based JavaScript component for viewing large data tables within a Web browser, reformatting XML VOTables into HTML in the browser.

OpenCADC-MongoDB Interface

After studying NoSQL DBMS and showing their pros, in this section we assume we want to rewrite the Web application from ASA shown in Fig. 4.4 to use a non-relational DBMS like MongoDB.

To take advantage of the existing OpenCADC work, our proposal is to use MongoDB and Java programming language.

We can connect Java and MongoDB using two methods:

1. the MongoDB-provided Java driver[39]; or
2. the Spring framework[40], a comprehensive programming and configu-

[38] http://code.google.com/p/voview/
[39] http://docs.mongodb.org/ecosystem/drivers/java/
[40] http://www.springsource.org

ration model for Java-based enterprise applications. The Spring Data MongoDB [41] provides an extension to the Spring programming model that support writing applications for the new datastores, like MongoDB or CouchDB database. There is no need to use any additional component, provided we have this software.

We have chosen the second option, as the Spring Data framework reduces notably the amount of code to be written. Let's see a comparison between the traditional Mongo way and the Spring Data one in Listing 4.7.

To accomplish this job, we have identified three major stages:

Defining a MongoDB schema for ObsCore To support the use cases identified in ObsCore [**?**], we must identify a set of elements that any ObsTAP service must support, ensuring that any query based on these parameters is guaranteed to be understood by all ObsTAP services. The ObsCore model must be implemented within TAP services, so that all queries can be executed, without changes have to be made, on any service that implements the model. `TAP_SCHEMA` is the database schema where the tables and columns exposed by the service are described.

MongoDB, thus, provides a very efficient way of supporting the optional columns of ObsCore by means of the variability of documents: particular attributes (ObsCore keywords) will exist or not depending on the document kind (FITS type), and any attribute can be missing, and still we can perform ObsCore queries: when a non-existing attribute is queried, is content is assumed to be NULL.

Ingesting data in the ObsCore database In Section 4.5 we already presented the model used to ingest the FITS files containing reduced data, and carrying ObsCore-compatible metadata.

For the ASA, FITS files contain many more columns than available in ObsCore. But that is precisely the strength of this approach: the additional keywords can be imported as document attributes, and queried against, without having to change the database schema, or change queries.

[41] `http://www.springsource.org/spring-data/mongodb`

Listing 4.7: Old vs new syntax Mongo-Java connection

```java
// The classic MONGODB-JAVA connection
public List<cols> getAll() {

  logger.debug("Retrieving all columns data");

  DBCollection coll =
      MongoDBFactory.getCollection("asa_db","coll_tables");
  DBCursor cur = coll.find();
  List<cols> items = new ArrayList<cols>();

  while(cur.hasNext()) {
    DBObject dbObject = cur.next();
    Cols cols = new Cols();

    cols.setColumnName(dbObject.get("Column_name").toString());
    cols.setTableName(dbObject.get("Table_name").toString());
    cols.setTableUcd(dbObject.get("Ucd").toString());
    cols.setTableUtype(dbObject.get("Utype").toString());

    ... // omitting the rest of fields

    items.add(cols);
  }

  return items;

}


// New implementation with Spring Data
public List<Cols> getAll() {

  logger.debug("Retrieving all columns data");

  Query query = new Query();
  List<Cols> cols = mongoTemplate.find(query, Cols.class);

  return cols;

}
```

Adapting the existing Java code The Java code that implements the OpenCADC interfaces needs to be rewritten to make use of MongoDB. For that, we need to:

- Implement the DataSource interface so that we can connect with the MongoDB instance.
- Implement the TapSchema class so that VO-metadata about the MongoDB attributes (data type, UCDs, UTypes, etc.) can be recovered from the MongDB attributes. A different document store is needed for that, where the documents represent the *database columns*, this is, the ObsCore document attributes themselves.
- All classes using SQL syntax need to use an adapter class that translates the SQL tokens into MongoDB operations. Fortunately, the jsqlparser tokenizer splits SQL in a parse tree that can be used for the reconstruction.
- Optionally, the QueryRunner class can be written using MapReduce jobs, instead of just using the SQL adaptor mentioned above, for further performance increases when doing queries.

We can see that the classes to be changed within the OpenCADC are a small subset. Notably, the classes dealing with the TAP endpoints, parsing of TAP queries, job management (UWS pattern), et cetera, remain untouched, and only the classes most directly involved with database interfacing need to be changed. That is partly thanks to the architecture of the OpenCADC, where classes are assembled through dependency injection, and responsibilities are clearly delimitated by the interfaces.

Chapter 5

Conclusions and future work

5.1 Conclusions

- Relational approach are not always suitable for any problem. Non-relational systems are not cure-all, but it has been shown they can face some problems in a more efficient way (*e.g.* MapReduce) and, in some situations, NoSQL can be a complement for existing RDBMS.
- NoSQL database systems, specially -not exclusively- those document-oriented can reduce system analysis and design and can also succeed in boosting the performance not just of data querying —their main goal—, but also simplify data management and data versioning, by simplifying ingestion and data updates.
- It is possible to convert an existing VO framework, such as the OpenCADC (in the configuration used for the ALMA Science Archive), without touching but a few classes. This gives VO tools access to NoSQL capabilities, without having to touch the user-exposed VO interfaces.

5.2 Future work

- Finish a reference NoSQL implementation of the OpenCADC.
- Use formal software engineering metrics (CoCoMo, Function Point Analysis, etc.) to evaluate the OpenCADC redesign and current and future costs involved.

- Benchmark the OpenCADC NoSQL system in order to obtain accurate data in performance improvements.