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

2013

Contents

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 (CERN), for instance, the Large Hadron Collider (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[1] (LOFAR), the Australian Square Kilometre Array Pathfinder[2] (ASKAP), the Large Synoptic Survey Telescope[3] (LSST), or the Square Kilometre Array[4] (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.

However, current facilities are also suffering from the lack of dedicated tools for astronomers, as their data and metadata sizes are also increasing beyond

[1] http://www.lofar.org
[2] http://www.atnf.csiro.au/projects/mira/
[3] http://www.lsst.org/lsst/
[4] http://www.skatelescope.org/

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[5] (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). 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.

[5] 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 include information 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 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 scalabilty 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 (150pc).

- Providing precise high dynamic range images at an angular resolution of 0.1*arcsec*

In order to accomplish those goals, ALMA will initially observe at wavelengths in the range $3mm$ to $400\mu m$ (84 to $720GHz$). The antennas can be moved around, in order to provide different baseline lengths. More extended arrays give high 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 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.

As stated in [10], 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 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 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.

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.
- 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 SKA:

- The data collected by the SKA in a single day would take nearly two million years to playback on an ipod.
- The SKA central computer will have the processing power of about one hundred million PCs.
- The SKA will use enough optical fibre to wrap twice around the Earth.
- The dishes of the SKA will produce 10 times the global internet traffic.
- The SKA will generate enough raw data to fill 15 million 64 GB iPods every day.
- SKA will be able to detect an airport radar on a planet 50 light years away.

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

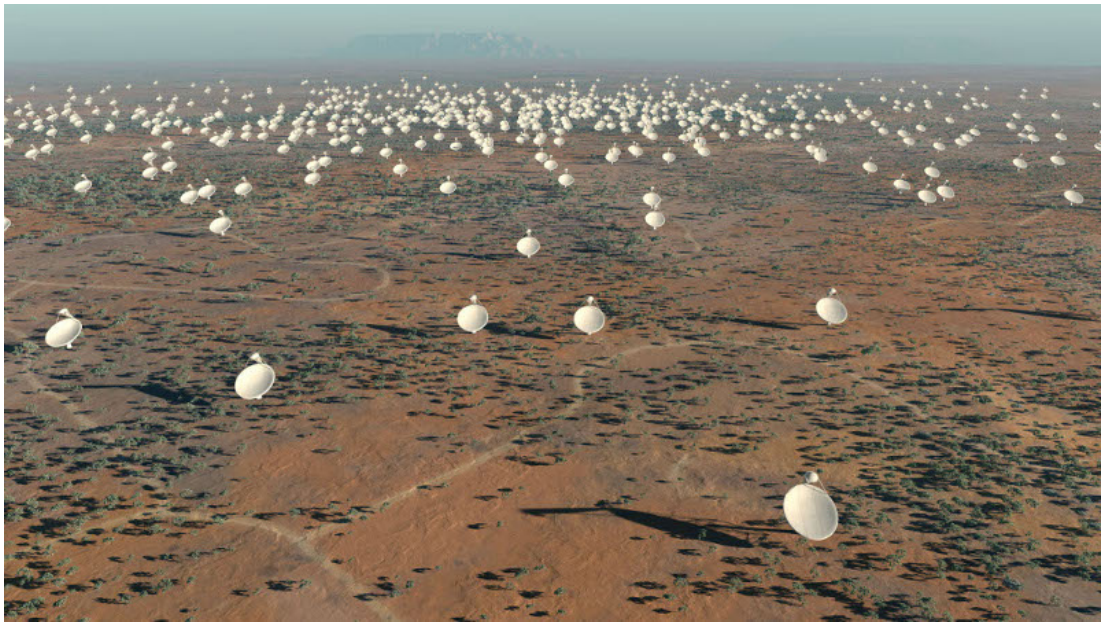


Figure 2.2: Artist's impression of the SKA dishes. Credit: SKA Organisation/T-DP/DRAO/Swinburne Astronomy Productions

2.3 Muchison Widefield Array

The Murchison Widefield Array is 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 isn not just a matter of storing the observations but how to distribute them from the MWA team in remote places, like MIT, Victoria University of Wellington in New Zeahland and India.

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

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

[4] http://www.skatelescope.org/news/pawsey-centre/

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, it can cover the sky twice a week. Data from 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 PC can fly through the Universe, zooming past objects a hundred million times fainter than can be observed with the human eye.

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

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] For a full list of institutional members, browse to `http://lsst.org/lsst/about/members`

Chapter 3

Successful NoSQL case studies

To support our proposal of implementing a NoSQL system, we present in this section some successful case studies which show that selecting such a new alternative (in comparison with the consolidated systems) to the classic RDBMS in big scientific projects is more than a risk, the main reason argued by NoSQL detractors.

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 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", so it can be deduced that this MongoDB approach is extensible.

3.2 ATLAS Workload Management System: Cassandra

A Thoroidal LHC Apparatus (ATLAS) is another of the six particle detectors experiments at LHC at CERN, whose data handling is a real challenge: almost 1 billion proton proton interactions per second 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[1] and Apache Hbase. As Cassandra has a lower learning curve, and considering that it was already deployed in ATLAS, the decision was done.

As a matter of fact, in the tests done, 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[2].

[1] More information at the Apache Cassandra Website, http://cassandra.apache.org/
[2] See http://cds.cern.ch/record/1446655/files/ATL-SOFT-PROC-2012-012.pdf

3.3 Measuring radiation levels in Seattle: CouchDB

Researchers at the University of Washington used the Cloudant [3] 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." 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.

[3] https://cloudant.com/

Chapter 4

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. 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 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.

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

4.1 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 and its Implementation in the Table Access Protocol [2]. That document was intended to be a description of the interface which integrated the data modeling and data access aspects in a single service: ObsCore data model + Table Access Protocol = ObsTAP.

4.1.1 ObsCore Components Data Model

Its aim is to get the generic data model for the metadata necessary to describe any astronomical observation. In the IVOA recommendation, the data are described using UML.

4.1.2 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. [3]

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

- TOPCAT
- TAPHandle which operates fully within the Web browser
- The TAP shell, a command line interface to querying TAP servers, complete with metadata management and command line completion.
- The GAVO VOTable library, which allows embedding TAP queries in Python.

[2] http://www.ivoa.net/documents/ObsCore/20110502/PR-ObsCore-v1.0-20110502.pdf

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

Listing 4.1: ADQL sample query

```
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)
```

The types of queries implemented by TAP are:

- Data queries
- Metadata queries
- Virtual Observatory Support Interface (VOSI)

TAP includes support for multiple query languages, including queries specified using the Astronomical Data Query Language (ADQL [1]) 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.

ADQL sample query:

4.2 Flexible Image Transport System

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 is 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.

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, 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.

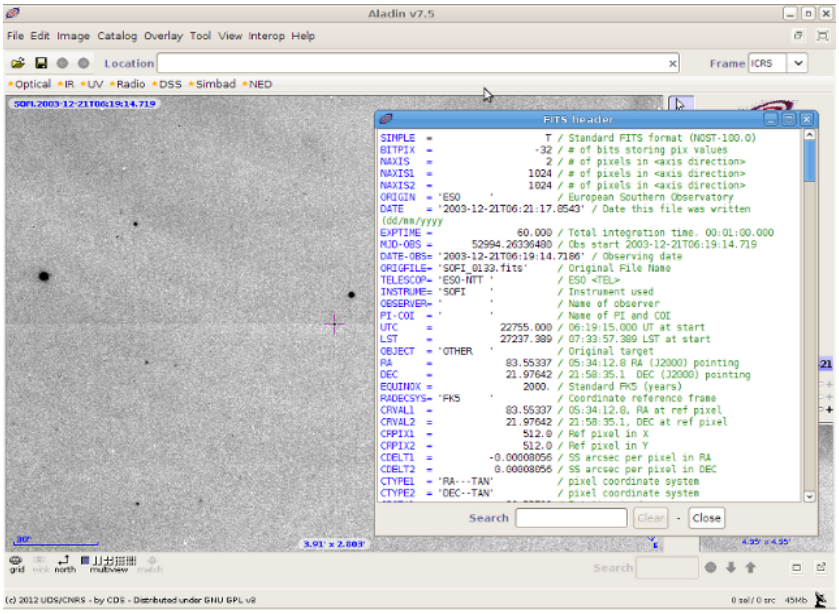


Figure 4.1: Viewing FITS header in Aladin

4.3 OpenCADC

OpenCADC (Canadian Astronomy Data Center) is a Virtual Observatory, used in ALMA Science Archive (for this reason is included in this chapter) tool which comprises several projects [4]:

4.3.1 Universal Worker Service

The Universal Worker Service (UWS) pattern defines how to build asynchronous (the client does not wait for each request to be fulfilled; if the client disconnects from the service then the activity is not aborted), stateful (the service remembers results of a previous activity), job-oriented (the rules for setting and arranging the parameters for a job is called Job Description Language- JDL) services.

When the execution starts, the job can adopt several states. The phases are the following:

Now, we make a subtle description of the structure of the code relating UWS in OpenCADC (the cadcUWS Library), the section of the code provides Job class and plugin architecture, servlet with UWS async and sync behaviour.

[4]We do not enumerate all of them here, for a full list, go to `https://code.google.com/p/opencadc/source/browse`

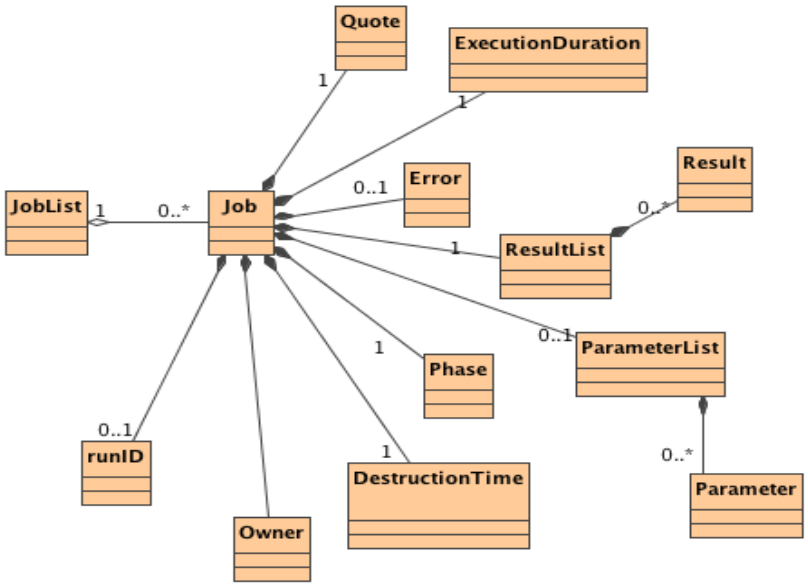


Figure 4.2: Class diagram representing UWS objects

JobManager

Responsible for job control.

JobPersistence

In charge of storing and retrieving Job state.

JobExecutor

It executes every job in separated threads.

JobRunner

The code that actually executes the job.

4.3.2 cadcTAP Library

The cadcTAP library is responsible of async and sync queries, where QueryRunner implements JobRunner. It also contains TAP_SCHEMA [5] Data Definition

[5]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

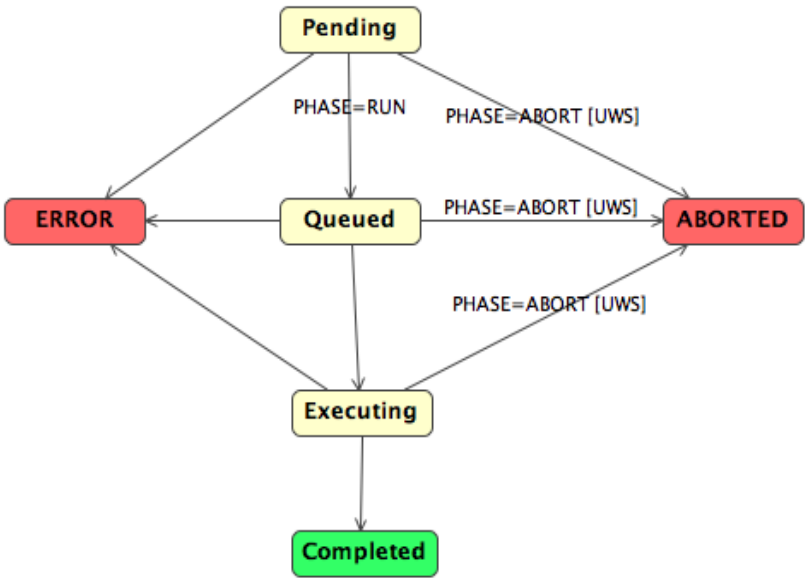


Figure 4.3: Universal Worker Server Job states

Language[6] (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 processess the query to local SQL.

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

- UploadManager

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

- TableWriter
- FileStore

QueryRunner

It implements JobRunner and sets Job state, find DataSource and uses TapSchema, UploadManager, TapQuery and TableWriter.

Chapter 5

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 performat. 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).

5.1 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 compliying 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
- BigTable
- Dynamo (Amazon)
- MongoDB
- Project Voldemort (LinkedIn)

Multivalue

- OpenQM

[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/

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
- db4o
- GemStone S
- Objectivity/DB

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

- HBase
- BigTable
- LevelDB (BigTable open version)
- Hypertable

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.

5.2 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 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 chosing 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.

Listing 5.1: 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
       }
);
```

5.3 MongoDB: a document oriented database

MongoDB (from "humongous") is an open source document-oriented database system developed and supported by 10gen. It is part of the NoSQL family of database systems. 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 5.1.

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.

5.3.1 Why MongoDB?

A more detailed list of features is listed in 5.3.2 , but as a summary, we have decided to use MongoDB for these reasons:

- 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 analytics platform [13], allowing to to send MongoDB data into Hadoop MapReduce jobs, process the data and return it back to a MongoDB collection).

5.3.2 Main features

Ad-hoc queries MongoDB supports search by field, range queries, regular expression searches. Queries can return specific fields of documents and 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[14]. The

[13] `http://hadoop.apache.org`

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

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 language itself.

5.3.3 The basics

We must know four concepts to understand MongoDB:

Database this concept is much like the RDBM counterpart.

Collection we can see a collection and a table as the same thing.

Document its equivalent in a RDBMS is the row; a document is made up of fields.

Field : it is equivalent to a column

Relational	Document oriented
Table	Collection
Row	Document
Column	Field

Table 5.1: Differences between relational and NoSQL terms

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 5.2 shows the differences in syntax between the DDL instructions for SQL and MongoDB.

5.4 FITS alternative 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 possible 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 much 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 [15], representing the users need for storing and handling the different information requirements:

- FITS Interferometry Data Interchange (FITS-IDI) Convention: conven-

[15]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

Listing 5.2: 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}
);
```

tions 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.

We can profit from MongoDB store units (collections, documents and fields). We could define as much 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 that shown on Listing 5.3 to create different variations of what a FITS document is like.

Now, inserting the primary HDU for a FITS-IDI header in MondoDB could not be easier, given the fact that FITS headers are collections of key-value pairs. We start from a very simple FITS header like:

```
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
```

Listing 5.3: MongoDB BSON code for creating the different FITS document types.

```
db.fits_super.insert(
  convention: "FITS-IDI"
);

db.fits_super.insert(
  convention: "SDFITS"
);

db.fits_super.insert(
  convention: "MBFITS"
);

db.fits_super.insert(
  convention: "OIFITS"
);
```

```
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
```

This code is very easy to translate into MongoDB syntax, as shown in Listing 5.4.

Now, we have our FITS header into a database with a document structure. We keep the same approach but in a very more usable and efficient way.

5.5 MapReduce in MongoDB

MapReduce, developed by Google, is a programming model and its implementation for processing (*e.g.* raw data from telescopes) and producing (*e. g.*

Listing 5.4: MongoDB BSON code for adding FITS metadata to the NoSQL database.

```
db.my_idi_collection.insert(
    supertype: "FITS-IDI", // it simulates a foreign key!
    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,
);
```

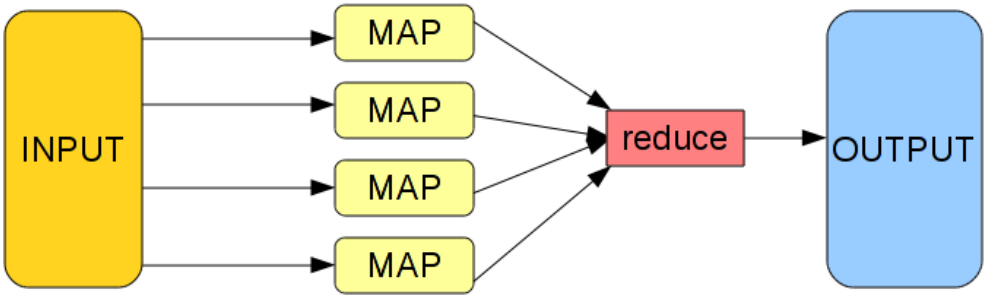


Figure 5.1: MapReduce concept

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.

MapReduce, usually, is used to solve problems involving big size datasets, up to several petabytes. For this reason, this model is used in distributed file systems, like HDFS [16].

The MapReduce model allows to write a map function which takes a key/-value pair, operates on it (performs object detection) and returns a new set of key/value pairs (source list). The reduce function then aggregates/merges all the intermediate data (builds an object catalog on a stacked source list). Many problems in astronomy naturally fall into this model because of the inherent parallelizability of many astronomical tasks. The concept is illustrated in Fig. 5.1.

To finish this chapter, we have developed a simple MapReduce job to show its use inside the context we are working on.

Listing 5.5: Example of MapReduce job

```
// 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"}
       }
)
```

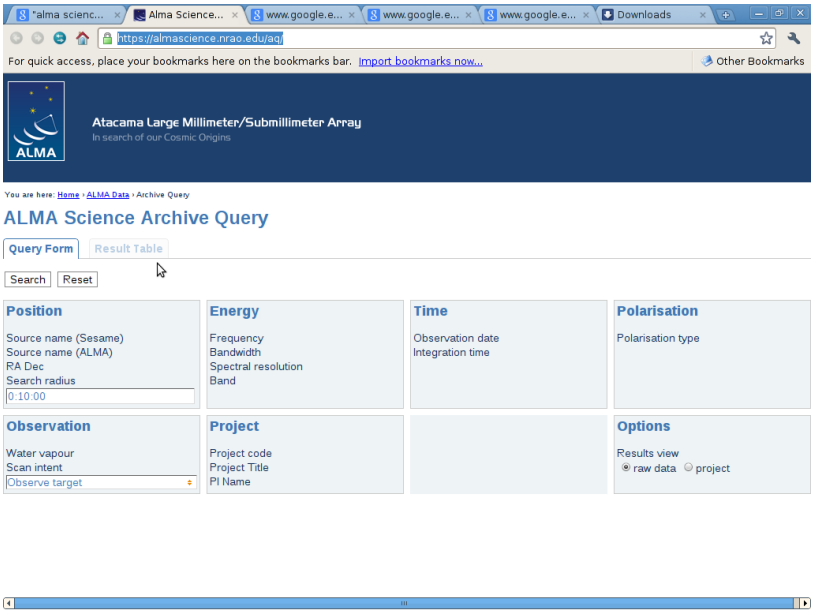


Figure 5.2: ALMA Science Archive Query

5.6 Rewriting the ALMA Science Archive with NoSQL

ALMA Science Archive (ASA) is the interface for querying ALMA data. 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, built as a web application.

The VO Technologies used in the ALMA Archive are (discussed in 4):

1. VO Data Models
2. Software
   (a) OpenCADC
   (b) VOView: a utility for viewing large data tables within a Web browser, reformatting a table in XML into HTML requested by the browser.

[16]For a full specification of this filesystem, go to `http://hadoop.apache.org/docs/stable/hdfs_design.html`

Java and MongoDB in ASA

Let's see some simple examples from ALMA query interface:

```
/* code from TapSchemaDAO.java */

// SQL to select all rows from TAP_SCHEMA.colums.
private static final String SELECT_COLUMNS_TEMPLATE =
  "select table_name, column_name, description, utype, ucd, unit,
     datatype, type_size " +
  "from %s.asa_columns ";

// List of TAP_SCHEMA.columns
List<ColumnDesc> columnDescs = jdbc.query(
  String.format(SELECT_COLUMNS_TEMPLATE, schemaName),
  new ColumnMapper()
);
```

Chapter 6

Conclusions and future work

6.1 Conclusions

- Data generated by huge scientific projects are becoming a problem.
- 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.
- Any new proposal should be inside Virtual Observatory frame.
- NoSQL database systems, specially -not exclusively- those document-oriented can reduce system analysis and design and can also succeed in boosting the performance of data management.

6.2 Future work

- Focusing in a workgroup inside Virtual Observatory instead of treating several aspects.
- The use of formal metrics (CoCoMo, Function Point Analysis, etc.) to plan the software design and development and the costs involved.
- Benchmarks support in order to obtain accurate data in performance improvements.

- Deciding which language to use to connect the selected DBMS.

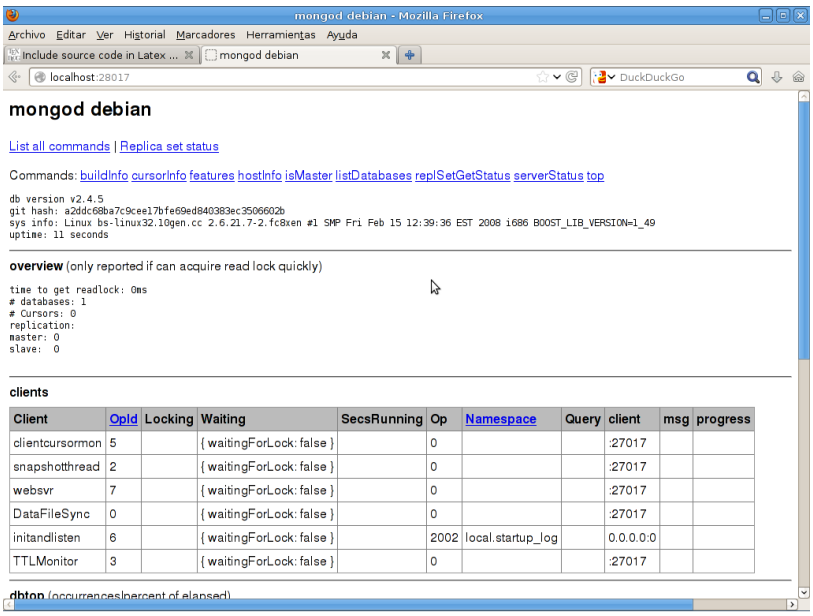
Appendix A

Getting and installing MongoDB

Go to `http://www.mongodb.org/downloads` and select our OS version. Download it and decompress. Supposing we are in a Linux box and that we are using the latest release (in June 2013) execute the following command:

```
./mongod --rest --dbpath /opt/mongodb-linux-i686-2.4.5/bin/data/db/
```

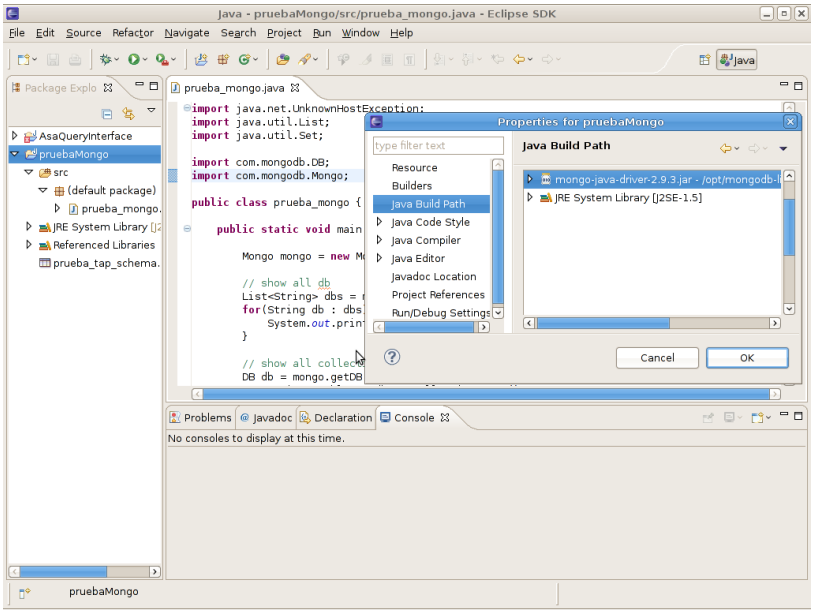
The server is now listening and waiting for connections, by default, in port 27017. As we have used the *–rest* modifier, we can point our browser to `http://localhost:28017` for http diagnostic access.



Appendix B

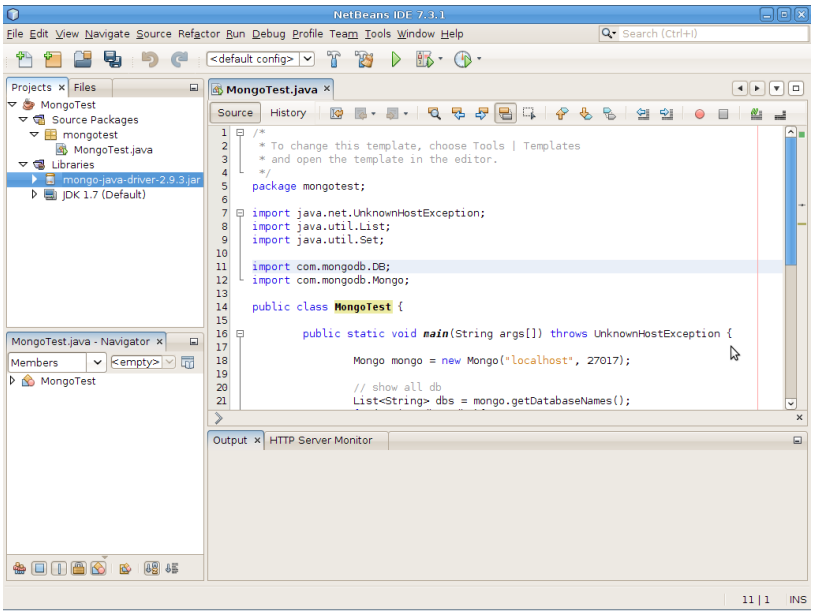
Configuring Eclipse and NetBeans for Java/Mongo development

- Go to http://central.maven.org/maven2/org/mongodb/mongo-java-driver/ and choose the right version.
- In Eclipse, just start a new project **File - New - Java Project**
- Right-click in **Properties** and then **Java Build Path - Libraries - Add External JARs**



- Import MongoDB methods, classed and interfaces as needed.
- In NetBeans, start a new project **File - New Project - Java Application**

- Right-click in **Libraries** and then **Add JAR/Folder**



- Import MongoDB methods, classed and interfaces as needed.

Bibliography

[1] Inc 10gen. The mongodb 2.4 manual. http://docs.mongodb.org/manual/, 2011-2013. [Online; accessed 01-December-2012].

[2] Kristina Chodorow. *MongoDB: The Definitive Guide*. O'Reilly, 2010.

[3] Kristina Chodorow. *50 Tips and Tricks for MongoDB Developers*. O'Reilly, 2011.

[4] Juan de Dios Santander Vela. Bases de datos multimedia para radioastronomía: Radams y dss-63. http://www.iaa.es/~jdsant/DEA-BBDDMultimediaParaRadioastronomia-RADAMS%2CDSS63.pdf, 2006. [Online; accessed 01-May-2013].

[5] Juan de Dios Santander Vela and Felix Stoehr. How the vo helped building the alma science archive. http://amiga.iaa.es/FCKeditor/UserFiles/File/VOandALMAarchive_web.pdf, 2012. [Online; accessed 26-April-2013].

[6] Juan de Dios Santander Vela et al. The alma science archive: Design. ftp://ftp.eso.org/projects/adass/posters/P147.pdf, 2011. [Online; accessed 27-April-2013].

[7] Juan de Dios Santander Vela et al. The alma science archive: Implementation. ftp://ftp.eso.org/projects/adass/posters/P133.pdf, 2011. [Online; accessed 07-February-2013].

[8] Abraham Silberschatz et al. *Database System Concepts Sixth Edition*. McGraw-Hill, 2010.

[9] Tony Hey et al. editors. The fourth paradigm. data-intensive scientific discovery. http://research.microsoft.com/en-us/collaboration/fourthparadigm, 2009. [Online; accessed 31-March-2013].

[10] Sandra Etoka. A first look at the alma science archive. http://www.eso.org/sci/facilities/alma/meetings/gar-sep07/pdf/Etoka.pdf, 2012. [Online; accessed 21-March-2013].

[11] Centre for Astrophysics and Supercomputing. Scientific computing and visualisation. http://astronomy.swin.edu.au/scivis/, 2013. [Online; accessed 30-January-2013].

[12] Centre for Astrophysics and Supercomputing. Accelerating astrophysics research with gpu supercomputing. http://www.sgi.com/pdfs/4401.pdf, 2013. [Online; accessed 30-January-2013].

[13] Survey Science Group. Putting astronomy's head in the cloud. http://ssg.astro.washington.edu/, 2013. [Online; accessed 7-July-2013].

[14] Nature Vol 455 — Issue no. 7209. Big data: science in the petabyte era. http://www.nature.com/nature/journal/v455/n7209/edsumm/e080904-01.html, 2008. [Online; accessed 14-February-2011].

[15] IBM Research. Square kilometer array: Ultimate big data challenge. http://asmarterplanet.com/blog/2013/03/the-square-kilometer-array-the-world%E2%80%99s-ultimate-big-data-challenge.html, 2013. [Online; accessed 04-July-2012].

[16] John Richer. Alma, synthesis imaging and the astro-grid. http://thames.cs.rhul.ac.uk/~fionn/astro-grid-papers/john-richer.pdf, 2001. [Online; accessed 22-November-2012].

[17] skatelescope.org. The age of astronomy big data is already here. http://www.skatelescope.org/news/pawsey-centre/, 2013. [Online; accessed 09-July-2012].