Universidad de Granada / Instituto de Astrofísica de Andalucía

BigData: Integrating NoSQL Technologies into Virtual Observatory

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

Chapter 1

Overview

The volume of data produced at some science centers presents a considerable processing challenge.

At CERN, for instance, almost 600 million times per second, particles collide within the Large Hadron Collider (LHC). Each collision generates particles that often decay in complex ways into even more particles. Electronic circuits record the passage of each particle through a detector as a series of electronic signals, and send the data to the CERN Data Centre for digital reconstruction. The digitized summary is recorded as a "collision event". Physicists must sift through the 15 petabytes or so of data produced annually to determine if the collisions have thrown up any interesting physics.

CERN does not have the computing or financial resources to crunch all of the data on site, so in 2002 it turned to grid computing to share the burden with computer centers around the world. The Worldwide LHC Computing Grid is a distributed computing infrastructure arranged in tiers which gives a community of over 8000 physicists near real-time access to LHC data.

So we could think that current deployed technologies are maybe obsolete and a new rethink should be made. If CERN (but not just CERN as we will discuss later) has changed its storage policy (leaving behind the classic client-server model), why not change the way the data are accessed?

Almost a decade after E. Codd published his famous relational model paper, the relational database management systems (RDBMS) have been the must-be tools. Today, non-relational, cloud, or the so-called NoSQL databases are gaining mindshare as an alternative model for database management. NoSQL movement is growing rapidly. Often more characteristics apply such as schema-free, easy replication support, simple API, eventually consistent / BASE (not ACID), a huge amount of data (big data) and more.

In this document, we present an alternative, using one of the NoSQL databases free available, to offer a different way of challenging these problems, and always operating inside VO frame.

Chapter 2

Astronomy 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, insight discovery and process optimization. Examples of big data include information such as Web search results, electronic messages (e.g. SMS, email and instant messages), social media postings, pictures, videos and even system log data. However, it can also include cash transactions, check images, receipts and other transactional information depending on the source of the information. This situation requires processing of terabytes and even petabytes of data. This is achieved by distributed processing. This is one of major reasons for the power of Web companies such as Google, Amazon or social networks like facebook and Twitter. Relational databases are found to be inadequate in distributed processing involving very large number of servers and handling Big Data applications.

Astronomical datasets are growing at an exponential rate: high performance computing applications in astronomy are enabling complex simulations with many billions of particles, while the forthcoming generation of telescopes will collect data at rates in excess of terabytes per day. This data deluge, both now and into the future, presents some critical challenges for the way astronomers derive new knowledge from their data.

In this chapter, we make an overview of some of the greatest telescopes, the way

they acquire and store data and the problem they face.

2.1 Atacama Large Millimiter Array

2.1.1 ALMA Instrument

ALMA is a worldwide project; the synthesis of early visions of astronomers in its three partner communities: Europe, North America and Japan. It is a fusion of ideas, with its roots in three astronomical projects: the Millimeter Array (MMA) of the United States, the Large Southern Array (LSA) of Europe, and the Large Millimeter Array (LMA) of Japan.

Figure 2.1: ALMA dishes in Atacama Desert

The origins of the Millimeter Array (MMA) are found in the science of the NRAO 36-Foot Telescope (later known as the 12-Meter Telescope), soon followed by the 4.9m telescopes at the University of Texas and Aerospace Corporation, the 14m telescope at the Five Colleges Radio Astronomical Observatory, and the 7m telescope at AT&T Bell Labs. Concerns with the limited size of the area available to the MMA on Mauna Kea and with potential environmental problems prompted a search for potential sites

in Chile. From April 1994, many highelevation sites were visited. Finally, the site retained for the MMA was formally proposed in 1996: it was the Chajnantor plateau.

This effort culminated in the signing of the ALMA Agreement on February 25, 2003, between the North American and European parties. More than 14 government agencies in Chile were involved in the negotiations.

Assuming all three partners are able to meet their commitments, it was decided that the final project would be cost-shared 37.5% / 37.5% / 25% between North America, Europe, and Japan, respectively. The observing time, after a 10% share for Chile, would be shared accordingly.

ALMA is an instrument that consists of a giant array of 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. Initially, it will observe at wavelengths in the range $3mm$ to $400m$ (84 to $720GHz$). The antennas can be moved around, in order to form arrays with different distributions of baseline lengths. More extended arrays will give high spatial resolution, 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), consisting of twelve 7-m antennas and four 12-m antennas. This array will mostly stay in a fixed configuration and is used to image large scale structures that are not well sampled by the ALMA 12-m array.

The design of ALMA is driven by three key science goals:

- The ability to detect spectral line emission from CO in a normal galaxy like the Milky Way at a redshift of $z = 3$, in less than 24 hours
- The ability to image the gas kinematics in protostars and in protoplanetary disks around young Sun-like stars in the nearest molecular clouds ($150pc$)
- The ability to provide precise high dynamic range images at an angular resolution of $0.1arcsec$

ALMA delivers data cubes, of which the third axis is frequency. In this sense, the final data products are very much like that of an integral field unit with up to a million Spectral Pixels.

2.1.2 ALMA Science Archive

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

- Persistent archiving and retrieval for observacional data.
- Observaction descrpitors.
- Datacubes produced by pipeline.
- Technical and environmental data.

And the key-point of the conceptual design of the ALMA Archive is to guarantee that three ALMA Regional Centres (ARCs) hold an identical copy of the archive at the Joint ALMA Observatory (JAO) in Santiago.

Relational denormalized database.

ALMA frontend archive is optimized for storage and preservation, not for data query and retrieval. ASA database is inspired from ObsCore, RADAMS and Hubble Legacy Archive plus Virtual Observatory Software:

- openCADC, which is used for database access and VO access protocol
- VOView, for Web components

2.2 Square Kilometer Array

Thousands of linked radio wave receptors will be located in Australia and in Southern Africa. Combining the signals from the antennas in each region will create a telescope with a collecting area equivalent to a dish with an area of about one square kilometre.

The Square Kilometer Array (SKA) will address fundamental unanswered questions about our Universe including how the first stars and galaxies formed after the

Big Bang, how galaxies have evolved since then, the role of magnetism in the cosmos, the nature of gravity, and the search for life beyond Earth.

The SKA is a global science and engineering project led by the SKA Organisation, a not-for-profit company with its headquarters at Jodrell Bank Observatory, near Manchester.

An array of dish receptors will extend into eight African countries from a central core region in the Karoo desert of South Africa. A further array of mid frequency aperture arrays will also be built in the Karoo. A smaller array of dish receptors and an array of low frequency aperture arrays will be located in the Murchison Radio-astronomy Observatory in Western Australia.

Construction is scheduled to start in 2016.

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

2.2.1 Figures and facts

The recent launch of the Murchison Widefield Array (MWA) - a radio telescope based in Western Australia's Mid West - marked the start of an impressive flow of astronomical data that will be stored in the iVEC-managed Pawsey Center in Kensington.

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". [1]

The Murchison Widefield Array is the first Square Kilometre Array precursor to enter full operations, generating a vast torrent of information that needs to be stored for later retrieval by researchers.

According to Proffesor Wicenec, "To store the Big Data the MWA produces, youd need almost three 1 TB hard drives every two hours'. The technical challenge isnt just in saving the observations but how you then distribute them to astronomers from the MWA team in far-flung places so they can start using it".

There are currently two links between the data stores in Perth and MWA researchers at the Massachusetts Institute of Technology (MIT) in the United States and the Victoria University of Wellington in New Zealand. A future link to India another MWA partner will also be created.

The data are not obviously intented to be fully available for everybody at everytime: 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.

MWA is producing so much information that it would be impossible to manually decide where to send what, which is where a sophisticated archiving system the

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

open-source Next Generation Archive System (NGAS) comes in. NGAS was initially developed by Professor Wicenec at the European Southern Observatory (ESO) and later modified by the ICRAR team to meet the MWA data challenge.

In the NGAS one simply asks the system for what the data wanted and it either provides it from the local store or retrieves it from the full archive back in Perth.

About half of all MWA computing occurs on site in the Murchison, where signals from radio telescope antennas are combined and processed in a powerful system of computers called a correlator. What's left to do in Perth is produce images, and manage storage and distribution by the archive system so MWA astronomers can analyze the collected data. Data travels down a dedicated 10 gigabit per second connection between the Murchison Radio-astronomy Observatory (MRO) and Geraldton, and the trip to Perth is completed on Australias new high-speed National Broadband Network.

The MWA will store about 3 Petabytes at the Pawsey Center each year. Another section of the Pawsey Center will be a supercomputing facility that includes computing for Australia's other SKA precursor, CSIROs Australian Square Kilometre Array Pathfinder (ASKAP), and projects from geoscience and other computationally intensive fields.

To sum up, 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.

- The SKA will be so sensitive that it will be able to detect an airport radar on a planet 50 light years away.

- Analysts estimate the London Olympics was the most data-heavy yet - with some 60 Gbytes, the equivalent of 3,000 photographs, travelling across the network in the Olympic Park every second. This however is only equivalent to the data rate from about half a low frequency aperture array station in SKA phase one.

2.3 Large Synoptic Survey Telescope

The Large Synoptic Survey Telescope (LSST) is a new kind of telescope whose wide field of view allows it to observe large areas of the sky at once. Taking more than 800 panoramic images each night, it can cover the sky twice each week. Data from LSST will be used to create a 3D map of the Universe with unprecedented depth and detail. This map can be used to locate the mysterious dark matter and to characterize the properties of the even more mysterious dark energy. Plans for sharing the data from LSST with the public are as ambitious as the telescope itself. Anyone with a computer will be able to fly through the Universe, zooming past objects a hundred million times fainter than can be observed with the unaided eye. The work on the project is broken down into three main areas: The Camera, Telescope and Site, and Data Management.

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 South America to the U.S.

Some of the institutional members are Google, Caltech, Harvard-Smithsonian

Center for Astrophysics, Fermi National Accelerator Laboratory or STSI. [2]

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

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

Chapter 3

The Virtual Observatory

The International Virtual Observatory Alliance (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." [1] The IVOA now comprises programs from Argentina, Armenia, Australia, Brazil, Canada, China, France, Germany, Hungary, India, Italy, Japan, Russia, Spain, the United Kingdom, Ukraine, and the United States and inter-governmental organizations (ESA and ESO). Membership is open to other national and international programs according to the IVOA Guidelines for Participation.

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 dis-

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

cuss 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.

3.1 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

3.1.1 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.

3.1.2 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

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

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.

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:

```
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,
```

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

```
 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 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, IGOR Pro, IDL, Java, LabVIEW, Mathematica, MatLab, Perl, PDL, Python, R, and Tcl. The FITS Support Office at NASA/GSFC maintains a list of libraries and platforms that currently support FITS.

Image processing programs such as ImageJ, GIMP, Photoshop, XnView and IrfanView can generally read simple FITS images, but frequently cannot interpret more complex tables and databases. Scientific teams frequently write their own code to interact with their FITS data, using the tools available in their language of choice.

The FITS Liberator software is used by imaging scientists at the European Space Agency, the European Southern Observatory and NASA. The SAOImage DS9 Astronomical Data Visualization Application is available for many OSs, and handles images and headers.

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.

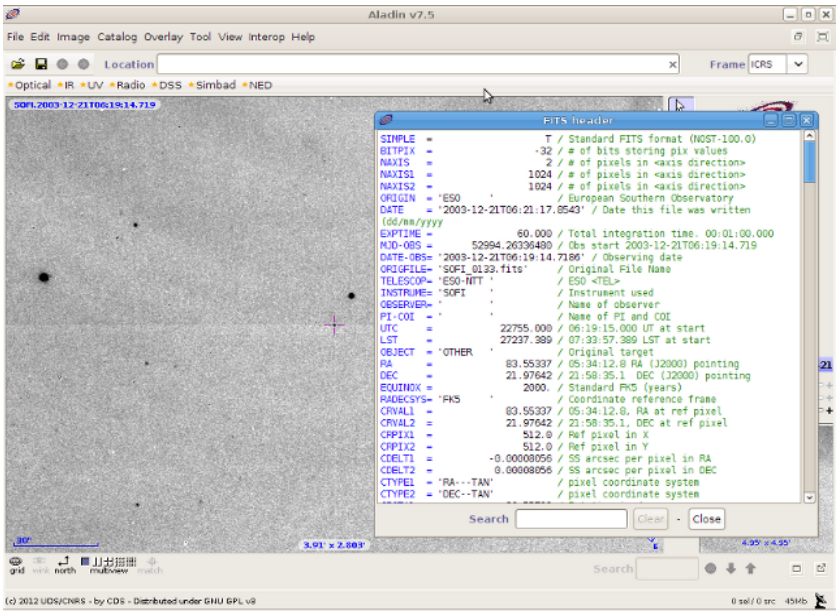


Figure 3.1: Viewing FITS header in Aladin

3.3 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]:

3.3.1 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.

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

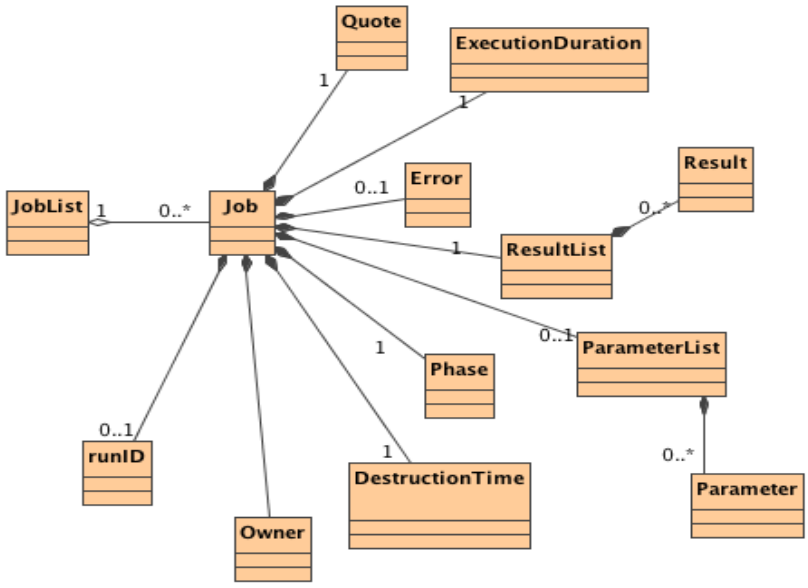


Figure 3.2: Class diagram representing UWS objects

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

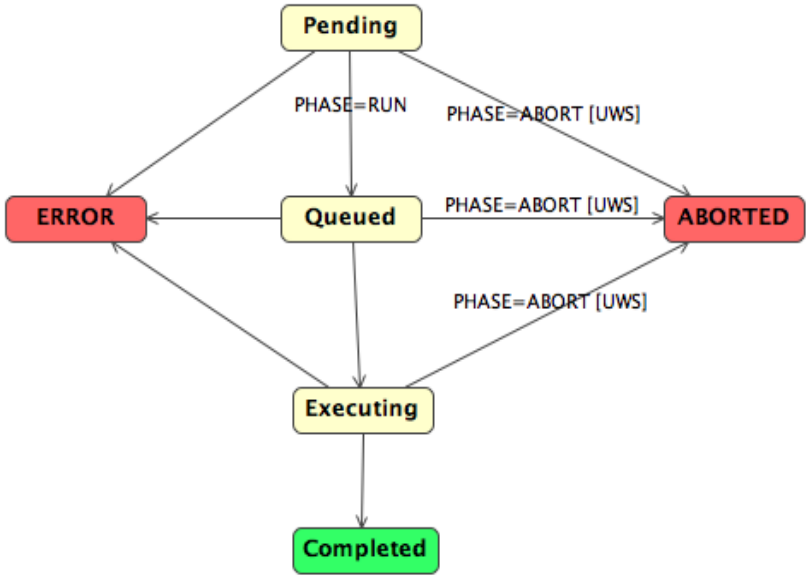


Figure 3.3: Universal Worker Server Job states

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.

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.

3.3.2 cadcTAP Library

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

[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

AdqlQuery

Same as before when $LANG = ADQL$.

Plugins

- UploadManager
- TableWriter
- FileStore

QueryRunner

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

Chapter 4

Non-relational DBMS inside VO

Typically, 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 can serve load lots of reads and writes. A non-relational database just stores data without explicit and structured mechanisms to link data from different buckets to one another.

IVOA has a standard called Astronomical Data Query Language (ADQL) that users do not necessarily need to know, because he 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 relational model. For the reasons stated in chapter 2, we propose the use of a different approach, the NoSQL technology.

4.1 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 green markers

(indicated to highlight the stories voted by others in the social network), the 6 TB of "ENSEMBLE" European Commission 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 ACID systems by adding an intermediate layer. 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, wit no extra overhead in data management, no system deployment middleware or any maintenance introduced by middleware layer.

Quite NoSQL systems employ a distributed architecture, maintaining data redundantly on multiple servers, often using distributed hash table. 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
  - CouchDB
  - MongoDB
  - RavenDB
  - IBM Lotus Domino
- Graph oriented
  - Neo4j
  - AllegroGraph
  - InfiniteGraph

  - Sones GraphDB
  - HyperGraphDB

- Key-value oriented
  - Cassandra
  - BigTable
  - Dynamo (Amazon)
  - MongoDB
  - Project Voldemort (LinkedIn)

- Multivalue
  - OpenQM

- Object Oriented
  - Zope Object Database
  - db4o
  - GemStone S
  - Objectivity/DB

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

They run on clusters of inexpensive machines.

4.2 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.

As an example, we show some JSON code inserting a document in a Mongo DB:

```
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
       }
);
```

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 features were excluded, namely 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

Relational	**Document oriented**
Table	Collection
Row	Document
Column	Field

Table 4.1: Differences between relational and NoSQL terms

entry, you can do so without affecting any other entry.

4.2.1 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 RDBMSes). Secondary indices are also available.

- **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.[9] The developer chooses a shard key, which determines how the data in a collection will be distributed. The data is split into ranges (based on the shard key) and distributed across multiple shards. (A shard is a master with one or more slaves.) 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** MongoDB could be used as a file system, taking advantage of load balancing and data replication features over multiple machines for storing files. This function, called GridFS,[10] is included with MongoDB drivers and

available with no difficulty for development languages (see "Language Support" for a list of supported languages). MongoDB exposes functions for file manipulation and content to developers. GridFS is used, for example, 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 kind of results for which the SQL GROUP BY clause is used.

- **Server-side JavaScript execution** JavaScript can be used in queries, aggregation functions (such as MapReduce), are sent directly to the database to be executed.

- **Capped collections** MongoDB supports fixed-size collections called capped collections. This type of collection maintains insertion order and, once the specified size has been reached, behaves like a circular queue.

  Once we have seen the main features of MongoDB, we can move on to the language itself.

4.2.2 The basics

We must know four concepts to dig into MongoDB's world:

- 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 RDBM is the row, and a document is made up of fields.

- Field: is a lot like a column.

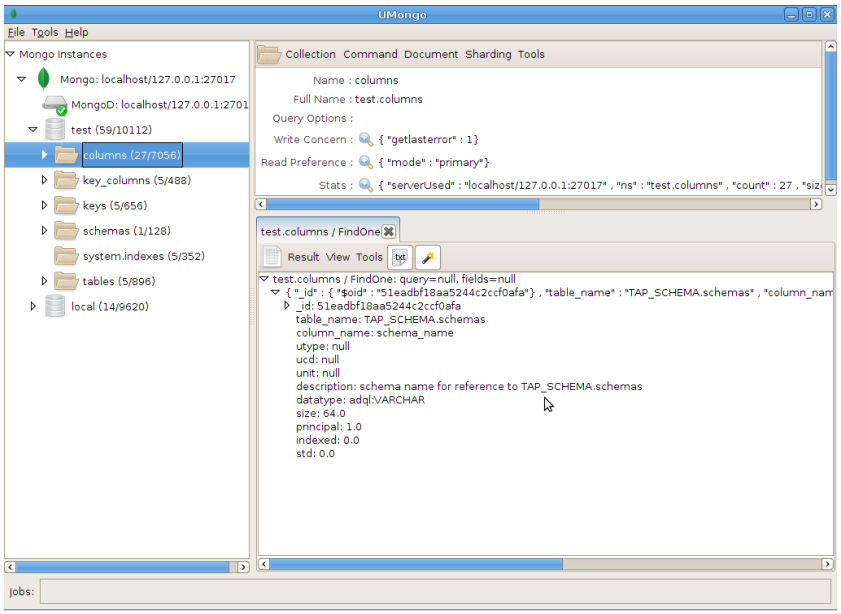


Figure 4.1: Tree View for Tap Schema in MongoVue

4.3 Advantages and uncertainties of NoSQL

- **Elastic scaling**

  For years, database administrators 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.

  RDBMS might not scale out easily on commodity clusters, but the new breed of NoSQL databases are designed to expand transparently to take advantage of new nodes, and theyre usually designed with low-cost commodity hardware in mind.

- **Big data**

  Just as transaction rates have grown out of recognition over the last decade, the volumes of data that are being stored also have increased massively. OReilly

has cleverly called this the industrial revolution of data. RDBMS capacity has been growing to match these increases, but as with transaction rates, the constraints of data volumes that can be practically managed by a single RDBMS are becoming intolerable for some enterprises. Today, the volumes of big data that can be handled by NoSQL systems, such as Hadoop, outstrip what can be handled by the biggest RDBMS.

- **No need for DBAs**

  Despite the many manageability improvements claimed by RDBMS vendors over the years, high-end RDBMS systems can be maintained only with the assistance of expensive, highly trained DBAs. DBAs are intimately involved in the design, installation, and ongoing tuning of high-end RDBMS systems.

  NoSQL databases are generally designed from the ground up to require less management: automatic repair, data distribution, and simpler data models lead to lower administration and tuning requirements in theory. In practice, its likely that rumors of the DBAs death have been slightly exaggerated. Someone will always be accountable for the performance and availability of any mission-critical data store.

- **Economics**

  NoSQL databases typically use clusters of cheap commodity servers to manage the exploding data, while RDBMS tends to rely on expensive proprietary servers and storage systems. The result is that the cost per gigabyte or transaction/second for NoSQL can be many times less than the cost for RDBMS, allowing you to store and process more data at a much lower price point.

- **Flexible data models**

  Change management is a big headache for large production RDBMS. 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 far more relaxed or even nonexistent data model restrictions. NoSQL Key Value stores and document databases allow the application to store virtually any structure it wants in a data element. Even the

more rigidly defined BigTable-based NoSQL databases (Cassandra, HBase) typically allow new columns to be created without too much fuss.

The result is that application changes and database schema changes do not have to be managed as one complicated change unit. In theory, this will allow applications to iterate faster, though,clearly, there can be undesirable side effects if the application fails to manage data integrity.

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 advocates will argue that their advancing age is a sign of their obsolescence, but for most CIOs, the maturity of the RDBMS is reassuring. For the most part, RDBMS systems are stable and richly functional. In comparison, most NoSQL alternatives are in pre-production versions with many key features yet to be implemented. Living on the technological leading edge is an exciting prospect for many developers, but enterprises should approach it with extreme caution.

- **Support**

  Enterprises want the reassurance that if a key system fails, they will be able to get timely and competent support. All RDBMS vendors go to great lengths to provide a high level of enterprise support.

  In contrast, 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 without the global reach, support resources, or credibility of an Oracle, Microsoft, or IBM.

- **Analytics and business intelligence**

  NoSQL databases have evolved to meet the scaling demands of modern Web 2.0 applications. Consequently, most of their feature set is oriented toward the demands of these applications. However, data in an application has value to the business that goes beyond the insert-read-update-delete cycle of a typical Web

application. Businesses mine information in corporate databases to improve their efficiency and competitiveness, and business intelligence (BI) is a key IT issue for all medium to large companies.

NoSQL databases offer few facilities for ad-hoc query and analysis. Even a simple query requires significant programming expertise, and commonly used BI tools do not provide connectivity to NoSQL.

Some relief is provided by the emergence of solutions such as HIVE or PIG, which can provide easier access to data held in Hadoop clusters and perhaps eventually, other NoSQL databases. Quest Software has developed a product Toad for Cloud Databases that can provide ad-hoc query capabilities to a variety of NoSQL databases.

- **Administration**

  The design goals for NoSQL may be to provide a zero-admin solution, but the current reality falls well short of that goal. NoSQL today requires a lot of skill to install and a lot of effort to maintain.

- **Expertise**

  There are literally millions of developers throughout the world, and in every business segment, who are familiar with RDBMS concepts and programming. In contrast, almost every NoSQL developer is in a learning mode. This situation will address naturally over time, but for now, it is by far easier to find experienced RDBMS programmers or administrators than NoSQL experts.

4.4 A FITS alternative with MongoDB

A very important 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).

Chapter 5

MongoDB in ALMA Science Archive

Chapter 6

Successful experiences

Chapter 7

Conclusions and future work

7.1 Conclusions

- Data generated by huge scientific projects are becoming a problem.
- Classical approaches are not always suitable for any problem.
- 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.

7.2 Future work

- It would be desirable to focus in a work group inside Virtual Observatory instead of treating several aspects.
- It is highly recommended to use formal metrics to measure the effectiveness of systems rewriting/redesigning.
- In order to obtain accurate data in performance improvements benchmarks must be used.

- It would be of interest to decide 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:

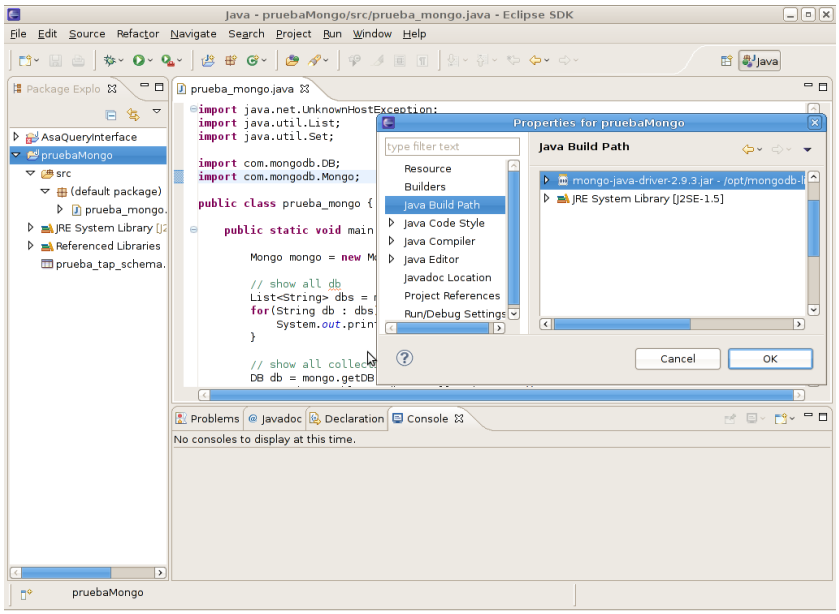
```
./mongod --rest --dbpath /opt/mongodb-linux-i686-2.4.5/bin/data/db/
```

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.

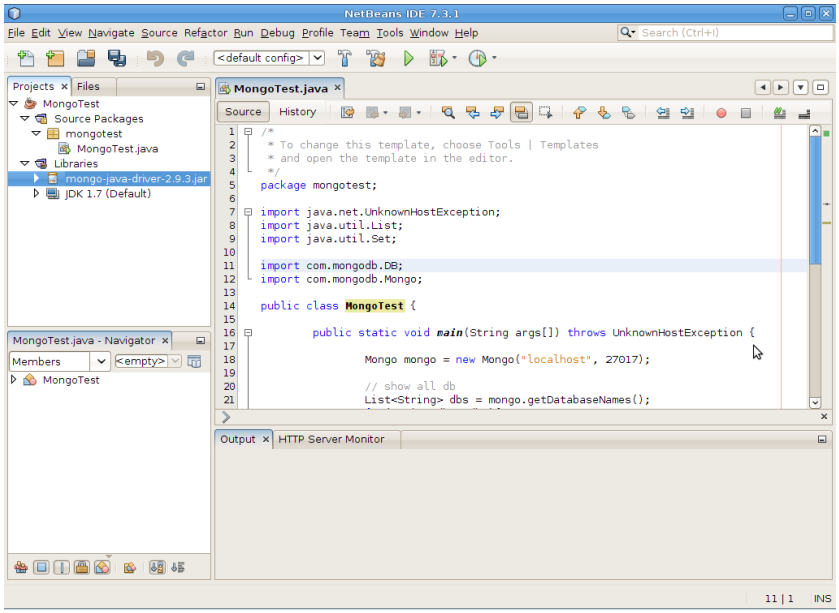
Appendix B

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.