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, several hundred million times per second, particles collide inside LHC. Each collision generates particles that often decay in complex ways into more particles. The pass of particles are recorded through a detector that sends the data for digital reconstruction. Physicists have then to browse through petabytes of data annually yielded to 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 its 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.

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 *de facto* tools. Today, non-relational, cloud, or the so-called NoSQL databases are growing rapidly as an alternative model for database management. Often more features apply such as schema-less, easy replication feature, simple API, eventually consistent / BASE (not ACID), a huge amount of data (big data) and more.

In this work, we give a short overview of big data problem and present an alternative, using one of the NoSQL databases (today, the volumes of data that can be handled by NoSQL systems exceed what can be handled by the biggest RDBMS) freely available, to offer a different way of challenging some of these problems, 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: the next generation of telescopes will collect data at rates in terabytes per day. This data deluge, now and int the future, presents some critical challenges for the way astronomers get new knowledge from their data.

In this chapter, we make an overview of some of the greatest telescopes and the amount of data they acquire and store.

2.1 Atacama Large Millimiter Array

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

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), 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 design of 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 $0.1arcsec$

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 producing a 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.
- Observaction descriptors.
- Datacubes produced by pipeline.
- Technical and environmental data.

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

ALMA front-end 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 (3.3), which is used for database access and VO access protocol
- VOView, for Web components

2.2 Square Kilometer Array

The Square Kilometer Array (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 official schedule) 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.

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

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.

2.3 Murchinson 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. [1]

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

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

techniques to control what data is copied from the Pawsey Center archive to the MIT machines are used. By 2013, more than 150 TB of data had been transferred automatically to the MIT store, with a stream of up to 4 TB a day increasing that value.

2.4 Large Synoptic Survey Telescope

The Large Synoptic Survey Telescope (LSST) is a new kind of telescope whose wide-field 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. [2]

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.

[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 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

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

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 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

[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.

- 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,
 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, 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

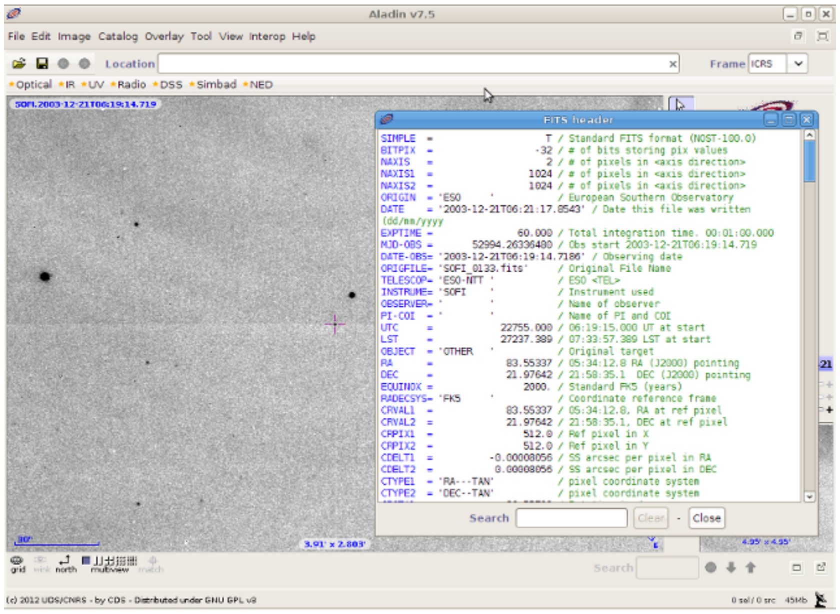
Each FITS file consists of one or more headers containing ASCII card images that carry keyword/value pairs, interleaved between data blocks. The keyword/value pairs provide information such as size, origin, coordinates, binary data format, free-form comments, history of the data, and anything else the 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.


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

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

Figure 3.1: Viewing 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]:

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

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.

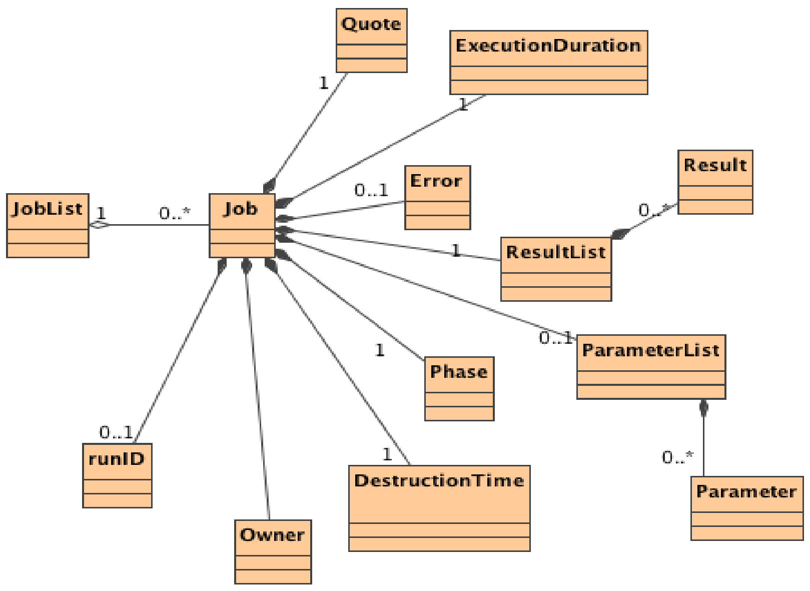


Figure 3.2: Class diagram representing UWS objects

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

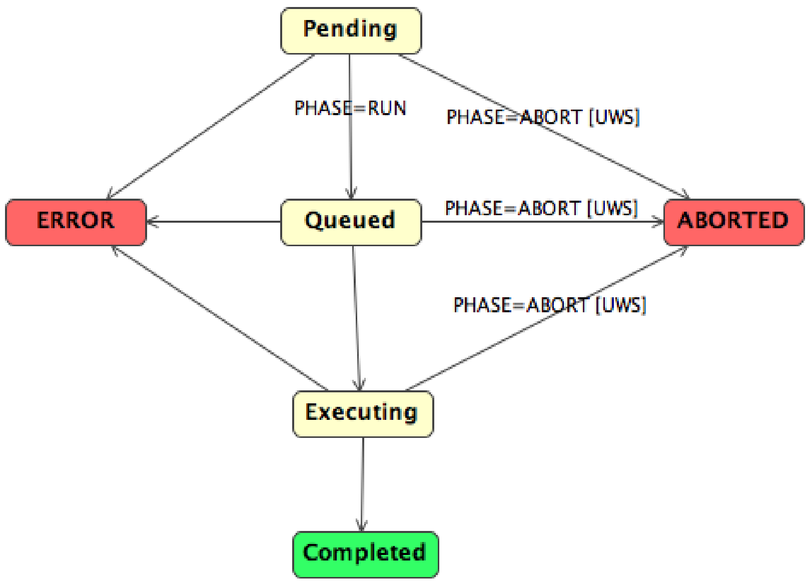


Figure 3.3: Universal Worker Server Job states

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).

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.

[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

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 DB are optimized for read and append 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.

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 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 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.

- **Expertise**

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

4.3 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.

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

4.3.1 Why MongoDB?

A more detailed list of features is listed in 4.3.2 , 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`).
- Widely used in the NoSQL community.
- It has full index support
- Very easy to install and to connect through drivers with several programming languages like Java, Python, Ruby, C/C++, PHP or Scala.
- It supports MapReduce paradigm.
- It is possible to connect MongoDB with Hadoop (the *de facto* big data processing and analytics platform [1], allowing to to send MongoDB data into Hadoop MapReduce jobs, process the data and return it back to a MongoDB collection).

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

4.3.2 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).

- **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. The data is split into ranges (based on the shard key) and distributed across multiple shards. MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure. Automatic configuration is easy to deploy and new machines can be added to a running database.

- **File storage** 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.

Relational	Document oriented
Table	Collection
Row	Document
Column	Field

Table 4.1: Differences between relational and NoSQL terms

4.3.3 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.

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):

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

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

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

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

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

-- DELETE

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

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

-- UPDATE

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

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

4.4 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 [2], representing the users need for storing and handling the different information requirements:

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

We show now a logical layout using MongoDB features to solve the previously

[2] 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

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, the code should be like this:

```
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"
);
```

Now, inserting the primary HDU for a FITS-IDI header in MondoDB could not be easier. 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
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:

```
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,
);
```

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.

4.5 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.* 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 [3].

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.

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

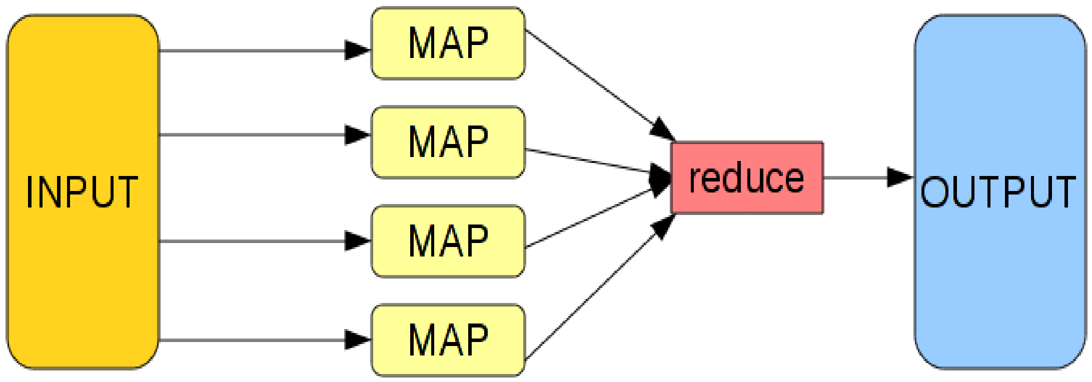


Figure 4.1: MapReduce concept

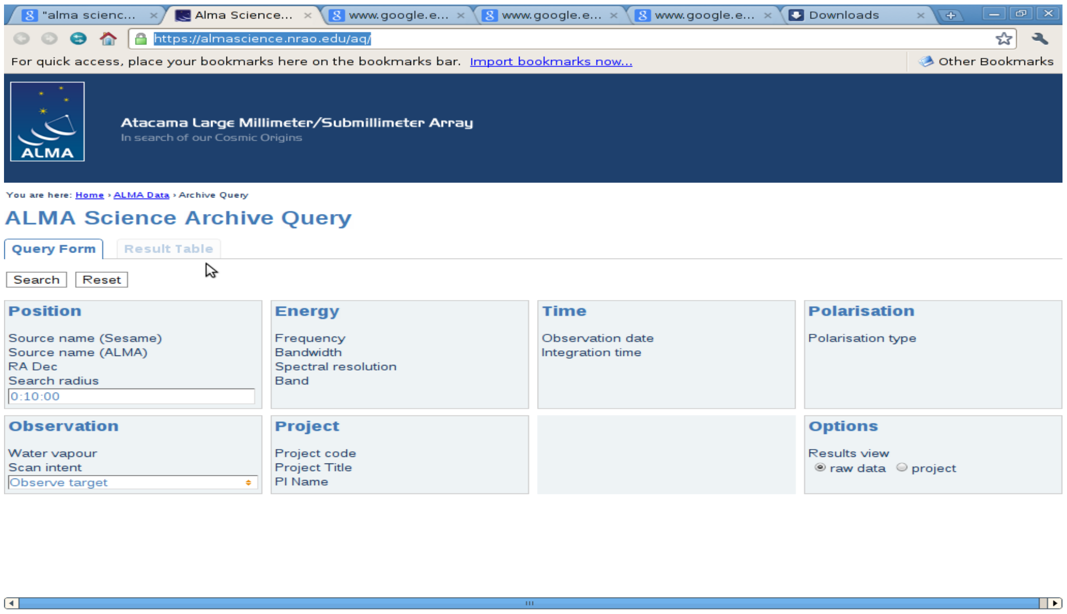
To finish this chapter, we have developed a simple MapReduce job to show its use inside the context we are working on.

```
// First, we define the map function

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

       emit({day: day, servername: this.servername},
               {count:1, timetaken: this.timetaken});
}

// Second, we define the reduce function

reduce = function Reduce(key, arr_values) {
       var result = {count: 0, countmore5: 0, countmore10: 0};
```

```
        for(var i in arr_values) {
                if(arr_values[i].timetaken > 5000) {
                        result.countmore5 += arr_values[i].count;
                } else if(arr_values[i].timetaken > 10000) {
                        result.countmore10 += arr_values[i].count;
                }
        }
        return result;
}

// Finally, we send these two functions to the server with the desided
//  filter, specifying where do we want the data to be sent

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

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

4.6 Rewriting ALMA Science Archive

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



https://almascience.nrao.edu/aq/

For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now... Other Bookmarks

ALMA

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

You are here: Home › ALMA Data › Archive Query

ALMA Science Archive Query

Query Form | Result Table

Search | Reset

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

Figure 4.2: ALMA Science Archive Query

web application.

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

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.

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 5

Successful case studies

5.1 CMS at the LHC

High-energy physicists working at the Compact Muon Solenoid (CMS) detector at the Large Hadron Collider (LHC) at Cern in Switzerland are benefiting from a NoSQL database management system that gives them unified access to data from a slew of sources.

Valentin Kuznetsov, a research associate and computer specialist at Cornell University, is a member of a team providing data management to the CMS Cern project. It built a system using MongoDB in preference to relational database technologies and other non-relational options.

“We considered a number of different options, including file-based and in-memory caches, as well as key-value databases, but ultimately decided that a document-oriented database would best suit our needs,” he says. ”After evaluating several applications, we chose MongoDB due to its support of dynamic queries, full indexes, including inner objects and embedded arrays, as well as auto-sharding.”

The CMS (pictured) is one of two particle physics detectors at Cern. It collects data from the LHC experiment that reproduces the big bang that kick-started the universe, designed to gain an understanding of how matter and force particles get their mass. More than 3,000 physicists from 183 institutions representing 38 countries are involved in the design, construction and maintenance of the CMS experiments.

Cornell is one of many institutions worldwide that contribute to the LHC experiments at Cern. Kuznetsov, a physicist and software engineer who has worked at Cern in the past, is involved in the data management group at Cern.

He says some five 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. It wanted to build a Google-like interface, but one that would return precise answers to queries whose form could not be determined in advance.

“We had a variety of distributed databases and different formats HTML, XML, JSON files, and so on. And then there is the security dimension. The complexity was, and is, huge”, he says.

The team discovered the world of NoSQL databases and, within that, the document-oriented approach seemed the best fit, with MongoDB being favoured because it lent itself well to the end product being an interface that could be queried in free text form.

MongoDB is one of a sub-genre of document store NoSQL databases, like Apaches CouchDB.

Barry Devlin, a data warehousing expert who blogs on TechTargets B-Eye Network, explains what is meant by a document-oriented approach: “Documents? If you’re confused, you are not alone. In this context, we don’t mean textual documents used by humans, but rather use the word in a programming sense as a collection of data items stored together for convenience and ease of processing, generally without a pre-defined schema.”

The CMS spans more than 100 datacentres in a three-tier model and generates around 10PB of data each year in real data, simulated data and metadata. This information is stored and retrieved from relational and non-relational data sources, such as relational databases, document-oriented databases, blogs, wikis, file systems and customised applications.

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.

The DAS provides a layer on top of the existing data sources that allows researchers and other staff to query data via free text-based queries, and then aggregates the results from across distributed providers, while preserving their integrity, security policy and data formats. The DAS then represents that data in defined format.

All DAS queries can be expressed in a free text-based form, either as a set of keywords or key-value pairs, where a pair can represent a condition. Users can query the system using a simple, SQL-like language, which is then transformed into the MongoDB query syntax, which is itself a JSON (JavaScript Object Notation) record, said Kuznetsov.

“Due to the schema-less nature of the underlying MongoDB back end we are able to store DAS records of any arbitrary structure, regardless of whether it’s a dictionary, lists, key-value pairs, and so on. Therefore, every DAS key has a set of attributes describing its JSON structure,” says Kuznetsov.

Kuznetsov says it is now clear that “there was nothing specific to the system related to our experiment”. The approach is extensible.

At Cornell, other groups, including in ornithology, are facing similar problems and have expressed interest. He stresses that analytics should be the main ingredient. The system should learn from what is being asked and be able to answer more questions in a free way.

“The beauty of MongoDB,“ he says, “is that the query language is built into the system,“ unlike, for example, Couch, also in use at Cern, where users need to code for each query.

”DAS is used 24 hours a day, seven days a week, by CMS physicists, data operators and data managers at research facilities around the world. The average query may resolve into thousands of documents, each a few kilobytes in size. [We get a] throughput of around 6,000 documents a second for raw cache population,“ concludes Kuznetsov.

The Cern physicists are at work unlocking the mysteries of the universe, and NoSQL technology is, whether they know that or not, helping them do it.

5.2 PanDA Workload Management System

PanDA is a Workload Management System built around the concept of Pilot Frameworks [1]. In this approach, workload is assigned to successfully activated and validated Pilot Jobs, which are lightweight processes that probe the environment and act as a smart wrapper for the payload. This 'late binding' of workload jobs to processing slots prevents latencies and failures in slot acquisition from impacting the jobs, and maximizes the flexibility of job allocation to globally distributed and heterogeneous resources [2]. The system was developed in response to computing demands of the ATLAS collaboration at the LHC, and for a number of years now has been the backbone of ATLAS data production and user analysis.

A central and crucial component of the PanDA architecture is its database (hosted on an Oracle RDBMS), which at any given time reflects the state of both pilot and payload jobs, as well as stores a variety of vital configuration information. It is driven by the PanDA Server, which is implemented as an Apache-based Python application and performs a variety of brokerage and workload management tasks, as well as advanced pre-emptive data placement at sites, as required by the workload.

By accessing the central database, another PanDA component -the PanDA Monitoring System (or simply PanDA Monitor)- offers to its users and operators a comprehensive and coherent view of the system and job execution, from high level summaries to detailed drill-down job diagnostics. Some of the entities that are monitored in the system are:

Pilot Job Generators (schedulers) Queues (abstractions of physical sites) Pilot Jobs (or simply pilots) Payload Jobs (the actual workload)

All of these are related to each other by reference, i.e. an instance of the Pilot Job Generator is submitting pilots to a particular queue, a queue in turn can contain any number of pilots in various stages of execution, and payload jobs are mapped to successfully activated, running pilots. Payload jobs are also associated with their input/output datasets (details of data handling in PanDA go beyond the scope of this paper). The purpose of the PanDA Monitoring System is to present these objects and their logical associations to the user with optimal utility.

Characteristics of the data and current design of the PanDA monitor As already mentioned, the PanDA Monitor handles a variety of data, and perhaps the most important and widely accessed objects stored in its database are entries describing payload jobs. We used this type of data as the most relevant case study in considering our database strategies going forward. Each job entry is mapped onto a row in the Oracle table, with roughly a hundred attributes (columns).

The monitor is currently implemented as a Python application embedded in the Apache server. Database access is done via the cx_Oracle Python library. The Oracle server is handling loads of the order of 103 queries of varying complexity per second. There are more than 500,000 new entries generated daily, which results in multi-terabyte accumulated data load. Despite ongoing query optimization effort, there are capability and performance limits imposed by available resources of the current Oracle installation and nature of the queries themselves.

Since RDBMS features such as the ability to perform join operation across several tables can be costly in terms of performance, most of the widely accessed data stored in the PanDA Oracle database is de-normalized. For that reason, there is slight redundancy in the data and joins are not used in the application.

For performance and scalability reasons, the data stored in Oracle is partitioned into ”live” and ”archive” tables. The former contains the state of live objects (such as job being queued or executed in one of the clouds), while the latter contains a much larger set of data which is final (read-only, such as parameters of fully completed jobs) and has been migrated from the live table after a certain period of time after becoming static. In addition, we note that monitoring applications do not require guaranteed hard consistency of the data, which would be another crucial motivation to use a RDBMS. In other words, the time window between the instants when information is updated in the database by its client application, and when this is reflected in results of subsequent queries, is not required to be zero (which is achieved in RDBMS by making the write operation synchronous). Motivation for noSQL and choice of platform As explained above, we actually dont have compelling reasons to store the monitoring data in a RDBMS. In situations like this, a variety of so-called noSQL database solutions are often employed by major Web services that require extreme degrees of availability, resilience and scalability. Examples include Google, Amazon, Facebook, Twitter, Digg and many others, as well as brick and mortar businesses. In order to leverage this new technology, we initiated a R&D project aiming at applying this approach to elements of the PanDA monitoring system to make it future-proof and gain relevant experience going forward. The noSQL term applies to a broad class of database management systems that differ from RDBMS in some significant aspects. These systems include such diverse categories as Document Store, Graph Databases, Tabular, Key-Value Store and others. Based on the patterns observed in queries in PanDA, a conclusion was made that either a Key-Value or a Tabular Store would be an adequate architecture in the context of PanDA monitoring system. A significant portion of all queries done in PanDA is performed using a unique identifier each entry has, therefore representing a classic case of Key-Value query. Other types of queries are done by indexing relevant columns, which is a general approach that will still work with noSQL solutions which support indexing. We considered two of the most popular platforms: Apache Cassandra (Key-Value) and Apache Hbase (Tabular). The latter relies on Hadoop Distributed Filesystem. To cut on the learning curve, and for ease of operation, it was decided to choose Cassandra, which does not have such dependency. In addition, Cassandra was already deployed and undergoing evaluation by ATLAS personnel, giving us leverage in quickly acquiring our own R&D platform. Cassandra characteristics A quick description of Cassandra would be as follows: its a truly distributed, schema-less, eventually consistent, highly scalable key-value storage system. The key, in this context, can be of almost any data type, as Cassandra handles it as an array of bytes. The value is a collection of elements called columns (again, stored as arrays of bytes). The column is the smallest unit of data in Cassandra and consists of a name, value and a timestamp (used by Cassandra internally). This gives the developer flexibility in modeling objects in the data. Aggregation of columns with the same key is often called a row, and it can be thought of as a dictionary describing an object. A collection of rows pertaining to the same application is termed a column family and serves as a very distant approximation of what is called a table in the traditional RDBMS note however that Cassandra rows can be drastically different from each other inside a single column family, while rows in a RDBS table follow the same schema. By design, the write operation is durable: once a write is completed (and this becomes known to the client requesting this procedure), data will survive many modes of hardware failure. While an instance of Cassandra can be successfully run on a single node, its advantages are best utilized when using a cluster of nodes, possibly distributed across more than one data center, which brings the following features: Incremental and linear scalability capacity can be added with no downtime Automatic load balancing Fault tolerance, with data replication within the cluster as well as across data centers

Cassandra also features built-in caching capabilities which will not be discussed here for sake of brevity. The application is written in Java, which makes it highly portable. There are client libraries for Cassandra, for almost any language platform in existence. In our case, we chose Pycassa Python module to interface Cassandra. Data Design and migration scheme Design of PanDA data for storage in the Cassandra system underwent a few iterations which included storage of data as csv (comma-separated values) data chunks, bucketing of data in groups with sequential identifiers (to facilitate queries of serially submitted jobs) etc. After performance evaluation, a simple scheme was adopted where a single job entry maps onto a Cassandra row. The row key then is the same as used as primary index in the Oracle database, which is the unique ID automatically assigned to each job by the PanDA system. At the time of this writing, parts of the archived data in PanDA are mirrored on Amazon S3 system in csv format for backup and R&D purposes, grouped in segments according to the date of the last update. This body of data was used as input for populating a Cassandra instance as it effectively allows us to avoid placing an extra load on the production RDBMS during testing, is globally and transparently available and does not require DB-specific tools to access data (all data can be downloaded by the client application from a specific URL using tools like curl, wget etc). In this arrangement, a multithreaded client application (written in Python) pulls the data from Amazon, optionally caches it on the local file system, parses the csv format, and feeds it to the Cassandra cluster. Characteristics of the cluster used in the project In order to achieve optimal performance, Cassandra makes full use of the resources of the machine on which it is deployed. It is therefore not optimal to place a Cassandra system on a cluster of virtual machines, since there is nothing to be gained from sharing redundant resources. For that reason, after initial testing on a 4-VM cluster at CERN, a dedicated cluster was built at Brookhaven National Laboratory, with the following characteristics: 3 nodes with 24 cores, 48GB RAM and 1TB of attached SSD storage each. Its first version relied on rotational media (6 spindles per node in RAID0 configuration), but testing showed that it didnt scale up to performance levels of the PanDA Oracle instance, therefore storage was upgraded to SSDs. Main results of testing will be given below. The Oracle cluster used in PanDA had a total spindle count of 110 (Raptor-class rotational media). Its performance would doubtless benefit from using SSDs as well, but clearly at this scale this wouldnt be economical. Its worth noting that Cassandra use of resources is radically different between read and write operations the former is I/O bound, while the latter is CPU intensive. Depending on application, its important to provide sufficient resources in each domain. Monitoring load on the nodes as well as client machines that load and index data on the cluster is an important part of regular operations, troubleshooting and maintenance. We used the Ganglia system to accomplish that. Indexing Efficient use of data makes it necessary to create an effective indexing scheme in the data. Analysis of actual queries done against PanDA database shows that in addition to a simple row key query they often contain selection based in values in a few columns. This knowledge can be utilized in building a Cassandra-based application, by creating composite indexes mapped to such queries. This can be done in one of two ways: 1. Relying on native indexing in Cassandra 2. Creating additional lookup tables by a separate client application If one goes with the former, it effectively necessitates creation of composite columns in the column families. This is akin to duplication of parts of data and leads to inflation of disk space associated with the data. On the other hand, there is no extra code to be written or run against the database, as the cluster will handle indexing transparently and asynchronously. Producing indexes by hand (the second option) gives the operator complete control over when and how the data is indexed, and saves disk space since no extra columns are attached to column families. In the end, driven by consideration of referential integrity when retiring parts of data, it was decided to use the former option in creating indexes, trading disk space for ease of operation and maintenance. Indexes were designed as follows: we identified queries most frequently used in the PanDA monitor. As an example, there are queries that select data based on the following job attributes: computingsite, prodsourcelabel, date. A new column is created in each row, with a name that indicates its composite nature:

computingsite+prodsourcelabel+date. The value of such column may then look like "ANALY_CERN+user+20110601". This augmentation of data can be done either at loading stage, or by running a separate process at a later time, at discretion of the operator. When and if this column is declared as index for Cassandra (which is typically done via its CLI), the cluster will start building corresponding data structures, automatically and asynchronously. To utilize the index, client application must determine that the users request contains requisite query elements, and then a query is run on the cluster by selecting values of the composite column according to the users request. Seven indexes constructed in such a way cover a vast majority of all queries done in the Monitor. The remaining small fraction can be handled by using individual single column indexes in combination, since this is also allowed in Cassandra. It wont be as fast as a composite index since it will involve iterations over a collection of objects instead of a seek and fetch, however preliminary testing shows that the performance will still be acceptable, given relative scarcity of such queries. Results of testing A few client applications, written in Python, were created for purposes of data loading and indexing, as well as performance and scalability tests. The Cassandra cluster was populated with real PanDA data, and the data load was equal to one year worth of job records, covering the period of June 2010 to May 2011, in order to be approximately the same as in testing done with previous configurations. Based on observed query patterns, two main categories of performance and scalability tests were performed, along with comparisons with analogous queries against the original Oracle database: Random seek test, whereby a long series of queries were extracting individual data for randomly selected jobs from the past year data sample Indexed queries, when a number of representative complex queries were performed using composite indexes as described in the section above

In order to judge the scalability of the system, queries of both classes were run concurrently in multiple client threads, simulating real operational conditions of the database. Our capability to fully stress the cluster was limited to approximately 10 threads, by the available CPU resources on the client machine, however the results extracted provide useful guidance nevertheless. In the random seek test, Cassandras time per extracted entry was 10 ms, with 10 concurrent clients. This translates into 1000 queries per second, which is about twice the rate currently experienced by our Oracle database. An analogous test done against Oracle (from a Python client) yielded roughly 100ms per query. In the case of indexed queries, Cassandras time per entry was 4ms. This result is counterintuitive at first, when compared to the random query metric, because indexed queries result in additional disk operations (the index must be read before the actual data extraction). However this result makes sense if one considers that multiple rows extracted in the query were packaged more efficiently in the network layer during data transmission from the cluster to the client, i.e. effective overhead per entry was lower. A comparable Oracle test returned a range of values from 0.5ms to 15 ms. The queries that were run for comparison purposes against Oracle and Cassandra were effectively identical in scope and logic, i.e. full content of the row was pulled from the database in each case. When increasing the load further, we found that the Cassandra cluster will adequately handle loads of at least 1500 queries per second, which is approximately 3 times more than current query load in the PanDA Monitor. Actual timing results were less relevant because of the client machine limitations mentioned above, i.e. we were not able to reach stress limits of the Cassandra cluster at this time. Conclusions We identified a noSQL database system Cassandra as a promising technology platform to ensure scalability and performance of the PanDA monitor database, operating under conditions of consistently high data and query load. A few versions of data design and indexing were evaluated. Hardware requirements were confirmed via testing of the Cassandra cluster under realistic conditions and benchmarking it against Oracle RDBMS. A simple but effective technique of composite index creation was employed to guarantee high performance in most popular queries generated in the PanDA monitor. Quantitative scalability and performance test results are favorable and pave the way for integration of this new noSQL data store with PanDA monitor.

5.3 Measuring radiation levels in Seattle

Researchers at the University of Washington utilized the Cloudant NoSQL 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 (his other title is research associate professor of physics), studied particles captured from the five air filters at the universitys Physics and Astronomy building and used Cloudants CouchDB-based BigCouch database to store and process the data from its experiments.

The research team's shielded germanium detector and data acquisition hardware.

The teams results show radioactive particles first reached Seattle on March 17, but they were less abundant in volume and type than was expected based on research carried out after the Chernobyl disaster in 1986. In fact, the projects official website states, We stress that the overall amount of the radioactivity is extremely low, at least thousands of times below EPA limits. Miller attributes the difference in radiation levels resulting from Chernobyl to the fact that the reactor at Chernobyl was operating at full steam when it exploded, whereas his teams research suggests the Fukushima plants automatic safety system must have kicked in when the earthquake hit, turning off the reactor in the process.

Millers association with Cloudant certainly influenced his teams decision to use the product, but BigCouch which is more big-data-focused than its web-application-focused NoSQL counterparts is particularly well-suited for this type of job. According to Miller, the team created a MacGuyver-like setup in order to start monitoring radiation levels in a hurry. After installing the special air filters and sampling gigantic quantities of air, the trapped particles were run through a germanium detector to determine the unique fingerprints of isotopes created by nuclear fission. Miller says the general belief is that radioactive isotopes travel through the air attached to dust particles or concrete particles from the explosion.

Cloudants BigCouch database let the team keep up with a steady flow of data so it could process and analyze it, then share it with the various stakeholders in near-real-time. The team was changing the data about 20 times per day and writing complex workflows to process it, two tasks that fall into BigCouchs wheelhouse. The database has a built-in MapReduce engine to enable writing and processing the workflows, and it allows for secondary indices, which users can populate with new data from their MapReduce jobs and query very quickly. Miller actually helped create Cloudant in 2008 while doing post-doctoral research at MIT that involved analyzing huge data sets from CERNs Large Hadron Collider.

Although humans might not be at risk from the Fukushima-based radiation, their scientific experiments might not be so lucky. In an article by the University of Washingtons Office of News and Information, Millers colleague R.G. Hamish Robertson explained that the radiation levels can raise havoc with sensitive physics experiments. That includes one called Majorana, in which the UW physicists are deeply involved, that is being planned for a lab nearly [one] mile down in the proposed Deep Underground Science and Engineering Laboratory in the old Homestake Mine in Lead, S.D. Miller told me that project, which involves trying to determine what the universe is made of, has been suspended because the next step is to bring highly sensitive homegrown copper up from underground for machining, but that even the trace levels of radiation present in South Dakota could taint it and ruin the experiment.

The currently-available paper sharing the teams results and insights mentions some uncertainty based on questions about the particle sizes they were trying to capture, but Miller said his team has made progress on this front and that theyre very confident in the results. The team will release a final paper soon with more detail and a larger time frame of data points.

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.

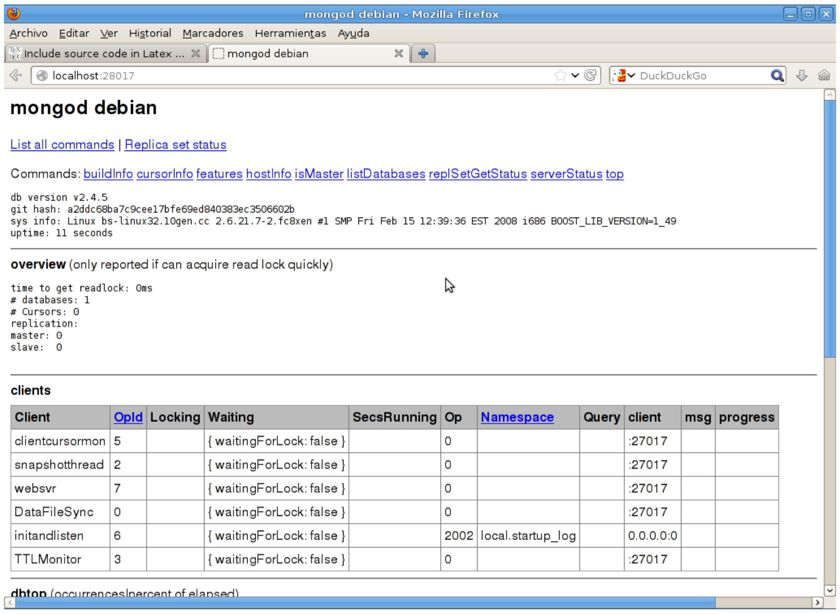
Appendix A

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.


mongod debian - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
Include source code in Latex ...
mongod debian
localhost:28017
DuckDuckGo

mongod debian

List all commands | Replica set status

Commands: buildInfo cursorInfo features hostInfo isMaster listDatabases replSetGetStatus serverStatus top

```
db version v2.4.5
git hash: a2ddc68ba7c9cee17bfe69ed840383ec3506602b
sys info: Linux bs-linux32.10gen.cc 2.6.21.7-2.fc8xen #1 SMP Fri Feb 15 12:39:36 EST 2008 i686 BOOST_LIB_VERSION=1_49
uptime: 11 seconds
```

overview (only reported if can acquire read lock quickly)

```
time to get readlock: 0ms
# databases: 1
# Cursors: 0
replication:
master: 0
slave:  0
```

clients

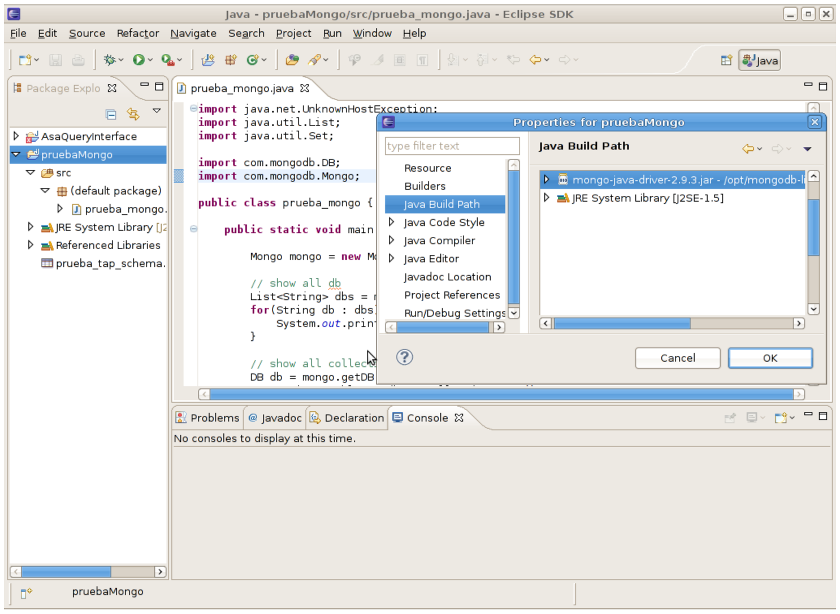
Client	OpId	Locking	Waiting	SecsRunning	Op	Namespace	Query	client	msg	progress
clientcursormon	5		{ waitingForLock: false }		0			:27017		
snapshotthread	2		{ waitingForLock: false }		0			:27017		
websvr	7		{ waitingForLock: false }		0			:27017		
DataFileSync	0		{ waitingForLock: false }		0			:27017		
initandlisten	6		{ waitingForLock: false }		2002	local.startup_log		0.0.0.0:0		
TTLMonitor	3		{ waitingForLock: false }		0			:27017		

dbtop (occurrences|percent of elapsed)

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**



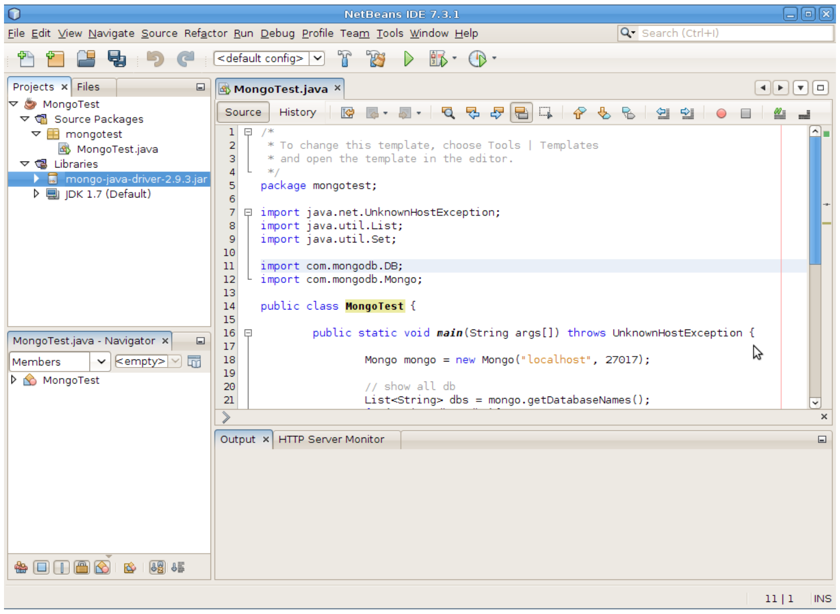
```
import java.net.UnknownHostException;
import java.util.List;
import java.util.Set;

import com.mongodb.DB;
import com.mongodb.Mongo;

public class prueba_mongo {

    public static void main

        Mongo mongo = new Mo

        // show all db
        List<String> dbs = r
        for(String db : dbs)
            System.out.prin
        }

        // show all collec
        DB db = mongo.getDB
```

Java Build Path

mongo-java-driver-2.9.3.jar - /opt/mongodb-l
JRE System Library [J2SE-1.5]

No consoles to display at this time.

- 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**



```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package mongotest;

import java.net.UnknownHostException;
import java.util.List;
import java.util.Set;

import com.mongodb.DB;
import com.mongodb.Mongo;

public class MongoTest {

        public static void main(String args[]) throws UnknownHostException {

                Mongo mongo = new Mongo("localhost", 27017);

                // show all db
                List<String> dbs = mongo.getDatabaseNames();
```

- 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].