Technology and Scientific File Repositories

Milena Ivanova, Martin Kersten, Stefan Manegold

Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands
M.Ivanova, Martin.Kersten, Stefan.Manegold@cwi.nl

Abstract. In this short paper we outline the *data vault*, a database-attached external file repository. It provides a true symbiosis between a DBMS and existing file-based repositories. Data is kept in its original format while scalable processing functionality is provided through the DBMS facilities. In particular, it provides transparent access to all data kept in the repository through an (array-based) query language using the file-type specific scientific libraries.
The design space for data vaults is characterized by requirements coming from various fields. We present a reference architecture for their realization in (commercial) DBMSs and a concrete implementation in MonetDB for remote sensing data geared at content-based image retrieval.

1 Introduction

Unprecedented large data volumes are generated by advanced observatory instruments and demand efficient technology for science harvesting [8]. To date, such data volumes are often organized in (multi-tier) file-based repositories. Navigation and searching for data of interest are performed using metadata encoded in the file names or managed by a workflow system. Processing and analysis are delegated to customized tools, which blend data access, computational analysis and visualization.

Wide adoption of DBMSs in scientific applications is hindered by numerous factors. The most important problems with respect to science file repositories are i) too high up-front cost to port data and application to the DBMS; ii) lack of interfaces to exploit standard science file formats; and iii) limited access to external science libraries for analysis and visualization.

To illustrate the problem we describe a concrete content-based image retrieval application over remote sensing images in the TELEIOS project[1]. The source data are high-resolution TerraSAR-X images in GeoTIFF format [7] accompanied by metadata specification in XML format. The processing pipeline starts with a preparatory phase of tiling an image into smaller chunks, called patches, and applying various feature extraction methods over the patches. The features extracted are stored in a database and used as input for higher level

[1] This work is partly funded by EU project TELEIOS: www.earthobservatory.eu

image analysis, such as classification with Support Vector Machines and content-based image retrieval. The DBMS is currently used at a relatively late stage of the pipeline to store the image features, which are of standard data types such as real numbers and strings. Both patching and feature extraction are carried out by customized software tools over image files in file repositories.

Ideally, a user could simply *attach* an external file repository to the DBMS, e.g. using a URI, and let it conduct efficient and flexible query processing over data of interest. However, a state-of-the-art relational database i) requires data to be loaded up-front in the database; ii) cannot naturally understand and support the external format of GeoTIFF images; and iii) provides limited processing capabilities for non-standard data types.

In this paper we outline the *data vault* that provides a true symbiosis between a DBMS and existing (remote) file-based repositories. The data vault keeps the data in its original format and place, while at the same time enables transparent (meta)data access and analysis using a query language. Without pressure to change their file-based archives, scientists can now benefit from extended functionality and flexibility. High level declarative query languages (SQL, SciQL [17]) facilitate experimentation with novel science algorithms. Scientists can combine their familiar external analysis tools with efficient in-database processing for complex operations, for which databases are traditionally good. Transparent, just-in-time load of data reduces the start-up cost associated with adopting a database solution for existing file repositories.

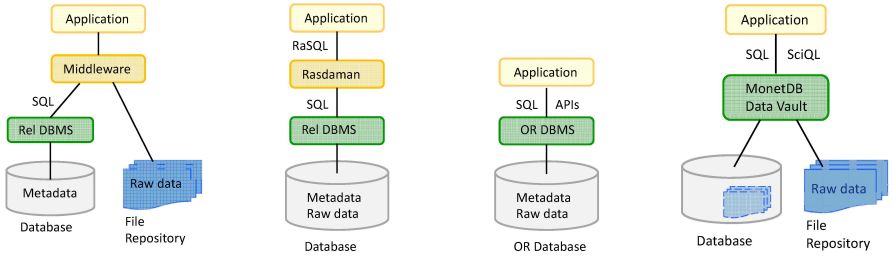
The data vault is developed in the context of MonetDB [11] and its scientific array-query language SciQL. It enables in-database processing of arrays, including raster images. In this work we focus on the data vault aspect and its realization in a real-world test case.

The remainder of this article is organized as follows. We summarize related work in Section 2. An analysis of general data vault requirements is presented in Section 3, followed by a description of our proposal for data vault architecture. Section 5 presents our work in progress on the design and implementation of a remote sensing data vault as a proof of the generic concept. Section 6 concludes the paper.

2 Related Work

Our work on data vaults is related to accessing external data from database systems. The implementation of a remote sensing data vault is a new approach to database support for external remote sensing data.

Access to external data. The SQL/MED standard (Management of External Data) [15] offers SQL syntax extensions and routines to develop applications that access both SQL and non-SQL(external) data. The standard has limited adoption for applications accessing data from other SQL-server vendors or in CSV files. NoDB [1] proposes advanced query processing over flat data files. Dynamic and selective load and indexing of external data provides low initialization overhead and performance benefits for subsequent queries. A



(a) File-based (b) Rasdaman (c) OR DBMS (d) Data Vault

Fig. 1: Software architectures for remote sensing data

major assumption of the above approaches is, however, that external data are straightforward mapped to SQL tables. In the domain of scientific applications we take a step further by considering array-based scientific file formats, such as GeoTIFF [7], FITS [6], and mSEED [14] which are typically used in Earth observation, astronomy, and seismology, respectively.

Oracle database file system (DBFS) [10] allows for storage of and access to files with unstructured content in the database. File processing is limited to a rudimentary, file system-like interface to create, read, and write files, and to create and manage directory structures.

Database support for remote sensing data. Figure 1 illustrates the development of software architectures for remote sensing applications and the role of the database management systems. Figure 1a presents a typical file-based solution where only the metadata are kept in the database (e.g. [16]). Such solutions have some important limitations, such as inflexibility wrt. new requirements, inefficient access, and scalability.

The use of database systems to open the raster data archives for scientific exploration is advocated in [2]. Figures 1b and 1c present architectures where the storage and retrieval of raster data (multi-dimensional arrays) is delegated to the DBMS. RasDaMan [3] array middleware provides applications with database services on multi-dimensional data structures, such as declarative query language, indexing, and optimization. It works on top of a DBMS which provides storage and retrieval for the array tiles and indexes in the form of BLOBs. However, the query processing is not integrated with the database internals.

Raster data management is also offered by some object relational DBMSs in the form of spatial data extenders: Oracle Spatial GeoRaster [12] and PostGIS [13]. They allow storing, indexing, querying, and retrieving raster images and their metadata. However, more complex analysis is delegated to the application or has to be implemented as a UDF. Images have to be loaded explicitly and up-front into the system to enable query processing over them. We propose just-in-time access to data of interest and white-box declarative queries over the sensor data itself. It has been shown that the declarative query paradigm increases productivity and offers flexibility that can be very valuable for ad-hoc science data exploration.

3 Data Vault Requirements

We can summarize the following requirements for the data vault design:

Enhanced data model. External file formats use their own specific data models to address the needs of a concrete problem domain. A core concept used in scientific formats is the array, for instance FITS, TIFF, and HDF5 all include arrays. Hence, the database model should be powerful enough to adequately represent the external concepts. We rely on a multi-paradigm data model in which both tables and arrays are first class citizens.

Repository metadata. The data vault opens up the file repository metadata, encoded in the self-descriptive files, for browsing and sophisticated searching by means of declarative queries. This allows fast identification of data of interest. It is important that access to the metadata is handled separately from the data, so that it does not require loading of the entire data set.

Just-in-time load. To deal with the shortcomings of up-front data ingestion, the data vault supports dynamic, just-in-time load driven by the query needs. This is justified by the usage patterns observed over scientific file repositories. It is rarely the case that all raw data i) have good quality and ii) are relevant for the current analysis. Often multiple versions of data, including low quality ones, are kept "just in case", but are never accessed in day-to-day operation. Consequently, there is no big use of putting the burden of low-quality or rarely used external data into the database.

Symbiotic query processing. The data vault enables several query processing alternatives. Data requested by a query can be loaded in the database and processed with pure database techniques. The data vault can also use external tools to process files in-situ and capitalize upon the existing support libraries. Symbiotic query processing can combine the benefits of both: use external tools, if efficient ones exist, and carry out operations in the database when the DBMS can perform them better.

A data vault should seamlessly integrate the available science libraries with database query processing. This requires extensions in the query optimization and processing layer of the database software stack. The query optimizer needs to distinguish operations that can be executed by external tools, and, based on a cost model, delegate such operations to be carried out in-situ over the original files.

Cache management. Once the data vault is set up, the scientists should be able to easily add to and modify files in the repository. To ensure correct functioning, the data vault requires users to refrain from making changes, such as deleting or moving the repository directory. The database management system commits itself to always present an up-to-date repository state to the user.

4 Data Vault Architecture

Next we present the design of the MonetDB [11] data vault architecture, illustrated in Figure 2. It extends MonetDB's software architecture with three components.

The *data vault wrapper* communicates with the external file repository and is built around *data model mapping*. It has components to access data, metadata, and external libraries and tools. The metadata wrapper takes care of accessing the metadata of the external file repository. It populates the *data vault catalog* in the database with a summary of the metadata to facilitate repository browsing, query formulation, and data management.

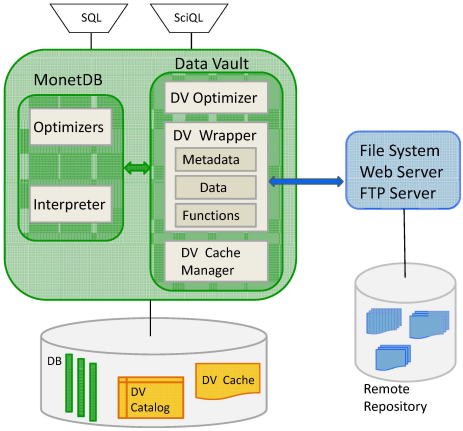


Fig. 2: MonetDB Data Vault Architecture

The data wrapper component accesses external data and creates their internal representation in the *data vault cache*. Similarly, data from the database can be exported to the external format and added to the repository. This facilitates reuse of existing tools to inspect the data products. The functionality wrapper provides access to external libraries. It defines mappings between external functions' input parameters and results to valid database representations.

The data vault cache contains a snippet of the repository data imported into the database. Subsequent analyses over the same external data will have it readily available without the need to repeat the (potentially expensive) importing transformations. This component leverages our work on recycling intermediates [9]. To enable caching of large data sets, the cache is augmented with mechanisms for spilling content onto the disk. The *data vault cache manager* is responsible for keeping the cache content in sync with the repository. The cached data can also be transformed into persistent structures in the database. This transformation can be workload-driven or induced by the user.

The purpose of the *data vault optimizer* is to enable symbiotic query processing. Using the data vault catalog, it detects operations over repository data and makes decisions about their execution locations based on a cost model. It might delegate processing to an external tool and correspondingly inject a call to the functional data wrapper. If in-database processing is expected to be more efficient, the optimizer will inject a call to the cache manager to provide a fresh copy of data under consideration. In turn, the cache manager can call the data wrapper for just-in-time loading of the external data, if they are not already available in the cache, or if the copy is outdated.

Implementation issues. Our architecture can be used for different data vault installations and applications. The data vault cache manager and query optimizer deal with generic tasks that are common for various data vaults. However, all wrapper components require separate treatment since they are specific for each file format and available software tools. There is an important trade-off between generality and application usability when it comes to the wrapper components implementation.

The most generic way would follow community-agreed specifications, often standards, for a particular data type and its operations. Examples are the object-relational extenders for spatial vector data. At the other end of the spectrum is a tailored implementation for a concrete format and application. An intermediate solution can follow a modular design where the common functionality is available as default modules, but the architecture is easily extensible with modules specific for the application.

To illustrate the problem, consider the design of the data vault catalog for remote sensing data in GeoTIFF format. The application at hand uses lots of metadata encoded in the file names and in auxiliary XML files. Only a small fraction of the standard GeoTIFF tags is used. The generic approach would present all metadata according to the format specifications, such as TIFF tags and geokeys. Hence, the application will have lots of needless metadata at hand, but may miss the ones encoded in the auxiliary sources. Consequently, the burden lies with the application developer to fill in the missing metadata. Alternatively, an application specific approach would limit itself to just the metadata needed. It would then, however, not be usable for other applications over the same external format. An implementation from scratch, including re-implementation of common functionality, is then called for.

5 Data Vault for Remote Sensing

In this section we describe ongoing work on the implementation of the data vault architecture for the remote sensing (RS) use case.

RS Catalog . The remote sensing data vault catalog stores metadata about the images as required by the application. The catalog is implemented as a set of relational tables in the `rs` schema. The `files` table describes the GeoTIFF files in the repository with their location, status (loaded or not), and the timestamp of the last modification. This table is generic for the file-based data vaults.

```
CREATE SCHEMA rs;

CREATE TABLE rs.files (
  fileid INT,
  location CHAR(256),
  status TINYINT,
  lastmodified TIMESTAMP );
```

```
CREATE TABLE rs.catalog (
  imageid INT,  fileid INT,  imagewidth INT,
  imagelength INT,  resvariant CHAR(4),  mode CHAR(2),
  starttime TIMESTAMP,  stoptime TIMESTAMP,
  sensor VARCHAR(20),  absorbit INT,
  PRIMARY KEY (imageid),
  FOREIGH KEY (fileid) REFERENCES rs.files(fileid) );
```

The `catalog` table describes image-specific metadata. We chose to explore an application-specific approach. Thus, the `catalog` table contains image metadata extracted from several sources: the metadata encoded in the GeoTIFF files (image length and width), the auxiliary data in the accompanying XML files (sensor), and properties encoded in the image file names (resolution variant, mode, etc.). The metadata wrapper is a procedure that takes as input parameter the absolute path name of the directory containing the image file repository, e.g., rs.attach('/data/images/geotiff'). It browses the directory, extracts the metadata encoded in different sources, and saves it into the RS catalog.

RS Data . Each image file can be represented in the system as a 2-dimensional array and queried with the proposed SciQL array query language [17]. The set of images available for database processing in the data vault cache is represented as a 3-dimensional array where the 3rd dimension is the associated imageid. The array is pre-defined as a part of the data vault and presents the data vault cache to the users so that they can formulate queries over images of interest in terms of it. However, the array is empty upon attachment of the repository as a data vault and images are not ingested up-front into the system.

```
DECLARE NumCols INT;   SET NumCols = ( SELECT max(imagewidth)  FROM rs.catalog );
DECLARE NumRows INT;   SET NumRows = ( SELECT max(imagelength) FROM rs.catalog );
CREATE ARRAY images (
  id INT DIMENSION,  x  INT DIMENSION [NumCols],  y  INT DIMENSION [NumRows],  v SMALLINT );
```

The data wrapper is responsible for the ingestion of external images when needed. It takes a set of image IDs as determined by the query criteria, locates the corresponding files, and loads them into the data vault cache structure, in this case the 3-dimensional array.

RS Query Processing . To clarify the query processing mechanism over remote sensing images from the data vault we start with a simple SciQL query. It computes image masks by simply filtering pixel values within the range [10,100]. The images are specified through predicates over the remote sensing catalog (e.g. the image resolution variant is spatially enhanced, imaging mode is High-resolution Spotlight, and the start time is in a given time interval).

```
SELECT [id], [x], [y], v   FROM images   WHERE v BETWEEN 10 AND 100   AND id in
(SELECT imageid   FROM rs.catalog
 WHERE resvariant = 'SE__' AND mode = 'HS' AND starttime > TIMESTAMP '2011-12-08 16:30:00');
```

The data vault optimizer recognizes the cache in the form of the `images` array and injects a call to the cache manager to provide the images as specified by the predicates over the `id` array attribute. The cache manager checks at run time if the images are available and issues a request to the data wrapper to ingest the missing ones into the `images` array.

We continue with an example content-based image retrieval application [5]. It takes an example image provided by the user and retrieves all images in the database similar to it according to some similarity measure. The measure used in our application is Fast Compression Distance (FCD) [4]. The main idea is to extract image dictionaries by applying some compression technique and to reason about similarities between two images based on the overlaps between their corresponding dictionaries. The dictionary can be extracted, for instance, with the LZW compression algorithm or by computing N-grams.

An example white-box SciQL function extracting 4-grams by row-wise processing of a 2-dimensional array is depicted below. It is then used to extract the 4-gram dictionaries from patches of a given image *imgid*.

```
CREATE FUNCTION dict4gram ( img ARRAY ( x INT DIMENSION, y INT DIMENSION, v SMALLINT ) )
RETURNS TABLE (dict_elem STRING)
BEGIN
  SELECT DISTINCT CAST(img[x][y  ].v AS STRING) || CAST(img[x][y+1].v AS STRING) ||
                  CAST(img[x][y+2].v AS STRING) || CAST(img[x][y+3].v AS STRING)
  FROM img   GROUP BY img[x][y:y+4];
END;

DECLARE patch_size INT;   SET patch_size = 256;
SELECT id, patch_size, x, y, dict4gram(v)   FROM images WHERE id = imgid
GROUP BY DISTINCT images[id][x:x+patch_size][y:y+patch_size];
```

The dictionaries can be stored in database tables and used to compute the FCD similarity measure as needed by the image retrieval application. The computation is illustrated with the following function that, given an image patch number and maximum distance, calculates the FCD and returns all patches with FCD smaller than the input parameter. We assume that the patch dictionaries computed above are stored in a table `imagedict(patch_id, dict_elem)`.

```
CREATE FUNCTION "FCD_1_n" ( pid INT, dist FLOAT )
RETURNS TABLE ( patch INT, distance FLOAT )
BEGIN
  DECLARE dict_size INT;
  SET dict_size = ( SELECT count(*) FROM imagedicts WHERE patch_id = pid );
  RETURN
    SELECT inter_size.patch AS patch,
      ( ( dict_size - inter_size.cnt ) / CAST( dict_size AS FLOAT ) ) AS distance
    FROM
      ( SELECT d2.patch_id AS patch, count(*) AS cnt
        FROM imagedicts d1 JOIN imagedicts d2 ON d1.dict_elem = d2.dict_elem
        WHERE d1.patch_id = pid
        GROUP BY d2.patch_id
      ) AS inter_size
    WHERE ( dict_size - inter_size.cnt ) / CAST( dict_size AS FLOAT) < dist;
END;
```

Expressing the processing steps of image patching, feature extraction, and similarity computation in SciQL and SQL offers greater flexibility for the user. For instance, it is easy to experiment how a given feature extraction method performs with different patch sizes by simply changing the value of the `patch_size` parameter and re-running the queries. Similarly, the system allows for convenient experimentation with and comparison between different versions of feature extraction methods and similarity measures.

6 Summary and Future Work

The data vault is a symbiosis between database technology and external file-based repositories. It keeps the data in its original format and location, while at the same time transparently opens it up for analysis and exploration through DBMS facilities. Scientists benefit from this functionality, flexibility, and scalability by combining external analysis tools with efficient in-database processing. Transparent, just-in-time load of data of interest reduces the start-up cost associated with adopting a pure database solution for existing file repositories.

A reference architecture to realize data vaults has been presented. A concrete implementation of the data vault using MonetDB has been described. It opens up a large archive of high-quality remote sensing radar images for data mining experiments.

The realization of the data vault provides a vista on different research challenges. Wrapping the external libraries functionality allows us to capitalize upon existing tools for in-situ analysis, but needs careful interface design and cost modeling. Efficient symbiotic query processing requires detection of external data and libraries and extensible cost model and optimizer.

Acknowledgments. We wish to thank our partners in the TELEIOS and COMMIT projects for constructive guidance on the functionality and implementation of the MonetDB Data Vault. In particular, we thank Y. Zhang for her important work on the implementation of SciQL.

References

1. I. Alagiannis, R. Borovica, M. Branco, S. Idreos, and A. Ailamaki. NoDB: Efficient Query Execution on Raw Data Files. In *SIGMOD*, 2012.
2. P. Baumann. Large-Scale Earth Science Services: A Case for Databases. In *ER (Workshops)*, pages 75–84, 2006.
3. P. Baumann et al. The multidimensional database system RasDaMan. *SIGMOD Rec.*, 27(2):575–577, 1998.
4. D. Cerra and M. Datcu. Image Retrieval using Compression-based Techniques. In *International ITG Conference on Source and Channel Coding*, 2010.
5. C. O. Dumitru, D. E. Molina, et al. TELEIOS WP3: KDD concepts and methods proposal: report and design recommendations. `http://www.earthobservatory.eu/deliverables/FP7-257662-TELEIOS-D3.1.pdf`.
6. FITS. Flexible Image Transport System. `http://heasarc.nasa.gov/docs/heasarc/fits.html`.
7. GeoTIFF. `http://trac.osgeo.org/geotiff/`.
8. T. Hey, S. Tansley, and E. K. Tolle. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, 2009.
9. M. Ivanova, M. Kersten, N. Nes, and R. Gonçalves. An Architecture for Recycling Intermediates in a Column-store. *ACM Trans. Database Syst.*, 35(4):24, 2010.
10. K. Kunchithapadam, W. Zhang, et al. Oracle Database Filesystem. In *SIGMOD*, pages 1149–1160, 2011.
11. MonetDB. `http://www.monetdb.org/`, 2012.
12. Oracle. Oracle Spatial GeoRaster Developer's Guide, 11g Release 2 (11.2).
13. PostGIS. `http://www.postgis.org/`.
14. SEED. Standard for the exchange of earthquake data, May 2010. `http://www.iris.edu/manuals/SEEDManual_V2.4.pdf`.
15. SQL/MED. ISO/IEC 9075-9:2008 Information technology - Database languages - SQL - Part 9: Management of External Data (SQL/MED).
16. E. Stolte, C. von Praun, G. Alonso, and T. R. Gross. Scientific data repositories: Designing for a moving target. In *SIGMOD Conference*, pages 349–360, 2003.
17. Y. Zhang, M. Kersten, M. Ivanova, and N. Nes. SciQL: Bridging the Gap between Science and Relational DBMS. In *IDEAS*, pages 124–133, 2011.