

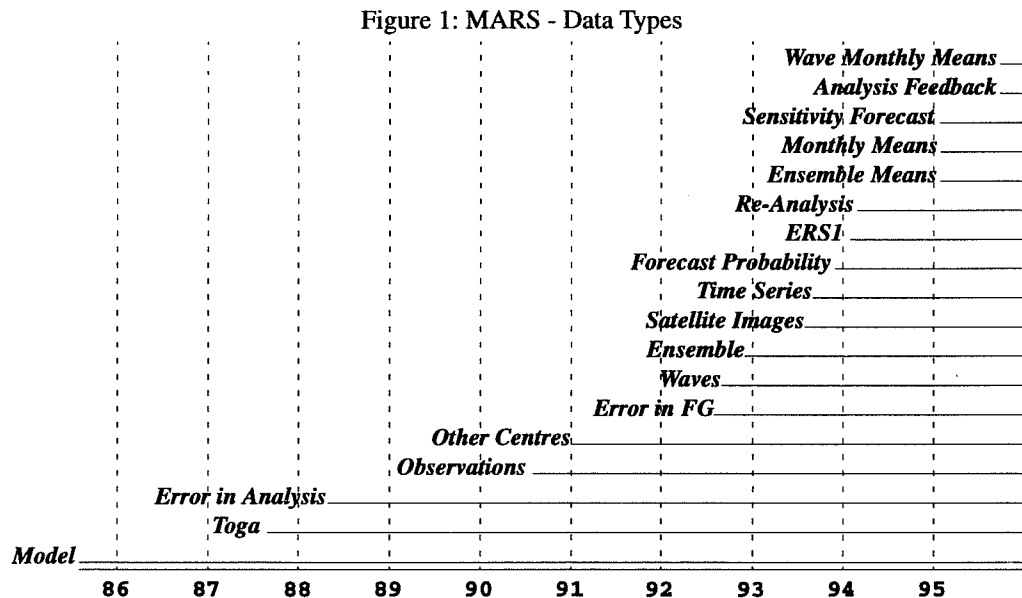
Architecture of the new MARS server

Baudouin Raoult
Meteorological Applications
ECMWF

1. Introduction

Since 1985, the ECMWF Meteorological Archival and Retrieval System (MARS) has grown both in size and diversity. At its start in 1985, the operational archive was growing at 70 Mbytes/day. In 1996, the growth rate was 125 Gbytes/day of operational data and 7 Gbytes/day of research data; the total archive was 22 Tbytes. The number of individual items archived is also very large: 120,000 operational fields/day, 140,000 reports/day and 200,000 research fields/day.

Figure 1 shows the various data types that are in MARS and the year in which they have been introduced. We actually have more data, for example we have had observations since 1979, but the data type "observation" was added to MARS in July 1990. Earlier data was then back-archived. Since this chart was drawn, new types have been added: 4D variational analysis, climatological simulations, seasonal forecasting, ensemble tubes and much more. As you can see, the curve shown is exponential, and we add new types more and more often.



We still have to add ocean models, coupled models, private datasets,... this list is endless.

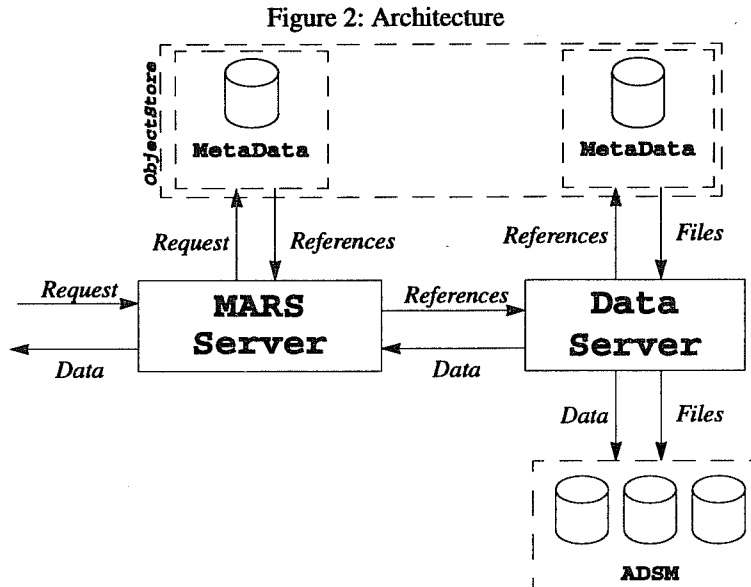
MARS is used in batch from the super-computers, interactively through Metview and remotely from the Member States. Every day, 30,000 requests are processed, 1,500,000 fields and 100 Gbytes are moved.

In order to cope with this growing archive and the ever changing requirements, a project was set up in 1996, called the DHS project (Data Handling System). MARS was totally redesigned to run on a Unix platform. It is written in C++ and uses ObjectStore to store its metadata and ADSM to store the actual data.

The new MARS had to be compatible with the old system. It has to provide a service for the next 10 years, and like every computer program, it has to be efficient, scalable and flexible. This paper describes how we are trying to achieve these goals.

2. Decoupling physical and logical organisation of the data

From the experience gained with the previous MARS systems, we knew we could build a scalable system if we could decouple the physical organisation of the data from its logical organisation. In order to do so, we split the system in two parts. The first part, the MARS Server, has semantic knowledge of the data. It knows what a meteorological field is, what a forecast is. The second part, the Data Server, has physical knowledge of the data. It knows if a piece of data is on tape, on disk or cached. Figure 2 shows the architecture of MARS.



The MARS Server does not handle files but data references. When a user request is processed, the MARS Server translates it into a list of data references that are passed to the Data Server. The Data Server translates data references into actual files and returns the data.

Using this design, we have a system that is independent from the underlying hardware and from the underlying software (ADSM). The data can be physically re-organised without any impact on the system. Data files are split or joined, moved from disk to tape without the need for the MARS Server to know. Something else we have learnt from the previous system is that the fewer files we have, the more manageable the system is. Most existing systems have problems managing more than a few million files. With this architecture, we reduce the number of files by merging them into larger files. Nowadays, we have more than 30 Tbytes in less than 200,000 files.

3. The metadata

The metadata is “data that describes the data”. In MARS, data are fields or observations. Metadata is data that knows where is which field or observation. This metadata is stored in an object oriented database, managed by a commercial product called ObjectStore. As we saw earlier, we need to split the logical organisation of the data from its physical organisation. This split is reflected in the metadata. In order to define this metadata, we first need to look at the data and answer the following question:

3.1 What is a meteorological field?

For MARS, a meteorological field is the smallest addressable object. It is defined by various attributes such as:

Attribute	Value
Class	Operational

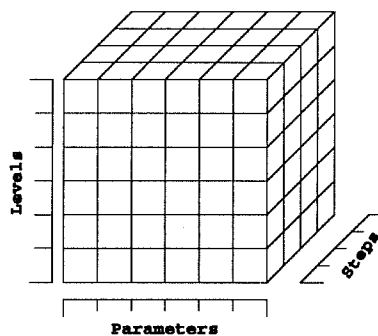
Attribute	Value
Version	1
Stream	Daily archive
Parameter	Temperature
Level	1000 hPa
Date	1993-08-10
Base Time	12Z
Time Step	120 H
Member	42

The attributes may be different for different data types. For example, the number attribute is only used to select a specific forecast number in an EPS (Ensemble Prediction System).

3.2 Archive objects

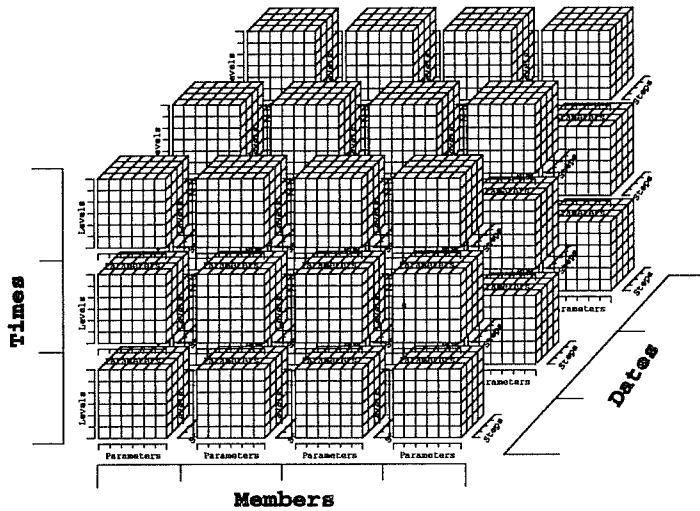
There are too many fields to store each individually. We need to group them into logical entities such as “a forecast”, “a month of analyses” or “a research experiment”. This provides a natural data co-location for a faster access to related data. These entities are called *archive objects*. We can stack all the fields for a forecast in a “cube” as shown in Figure 3. We group the fields using three attributes: level, parameter and step. This would represent a single forecast, for a given date, a given base time, and a given version. The three attributes are the three dimensions of the *archive object*.

Figure 3: Archive object



Because we group the fields using more than three attributes, the archive objects are actually hypercubes. Figure 4 is an attempt to draw a six dimension hypercube that would represent all the fields of all the forecasts of an EPS for a month.

Figure 4: Archive objects are hypercubes



3.3 Shapes and Layouts

We split the description of an *archive object* into its *shape* and its *layout*. The shape will be:

- Its meteorological content.
- Its logical organisation.
- Part of MARS metadata.

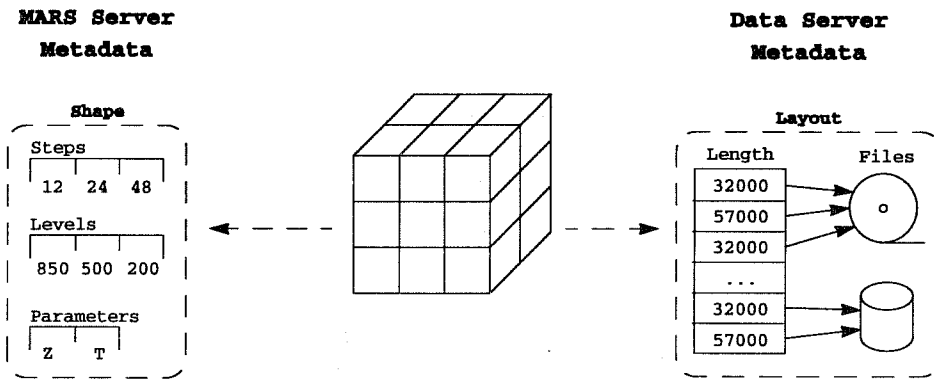
And its layout will be:

- Its physical organisation.
- Its physical location.
- Part of the Data Server metadata.

Figure 5 shows how we perform this split: the archive object is represented as a cube. The MARS Server metadata is simply a list of each dimension of the cube: in this case, three axes and their labels. This is called the “shape” of the archive object.

The Data Server metadata contains the size of each field, the file in which it is, and its offset in this file. This is the “layout” of the archive object. Each layout is given a unique identifier that is saved by the MARS Server with the shape.

Figure 5: Shapes and Layouts

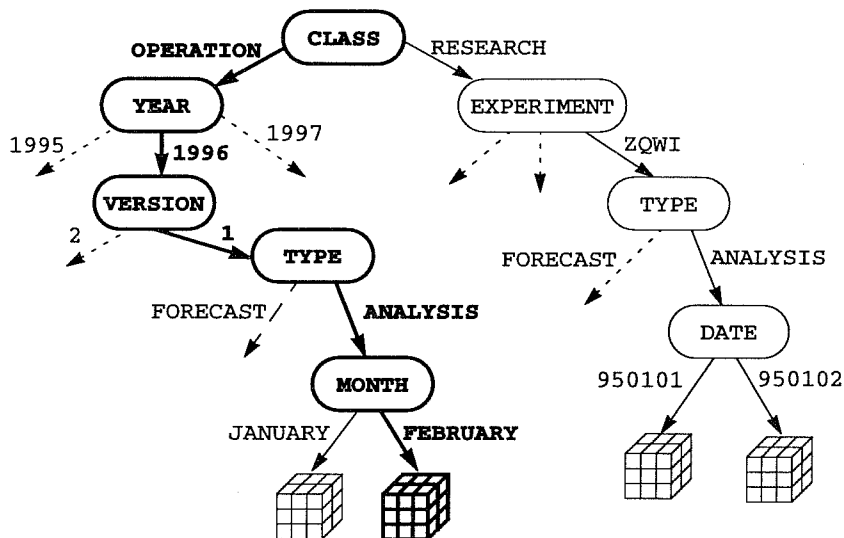


3.4 MARS Server metadata

All shapes are organised in a tree fashion, as shown in Figure 6. The nodes of the tree are defined by attributes that are not used to describe the dimensions of archive objects.

When a user request is received, it is used to navigate the tree. The highlighted path in Figure 6 corresponds to a request for some fields from the operational analysis of february 1996.

Figure 6: MARS Tree

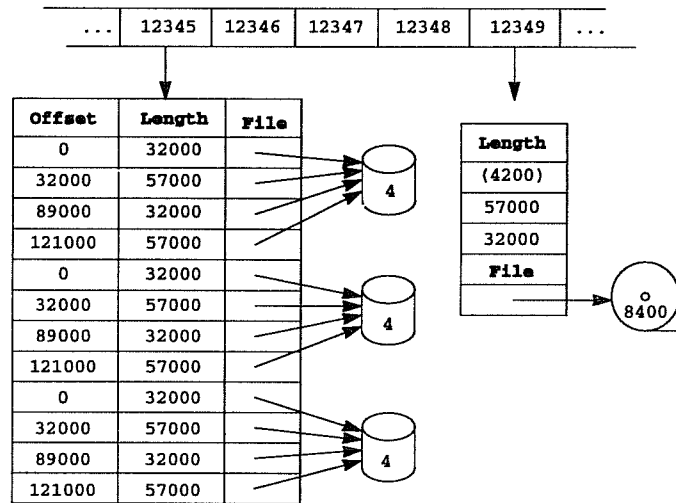


Because the design is object-oriented, each node and each shape can be different (polymorphism). New nodes can be added later, as new attributes are invented, and new shapes can be added later, as new data types are created. The branches of the tree can all be different and the tree can span several physical databases.

3.5 Data Server Metadata

The layouts are all organised in a large table, using their unique identifier as an index (Figure 7). A layout can be fully expanded, in this case it is itself a three column table. The first line correspond to the first field, the second line to the second field and so on. A layout can also be compacted. Most of the time, the fields have a fixed size, so we can use a simple run-length encoding scheme to reduce the size of the metadata. In Figure 7, the layout 12349 represents 8400 lines compacted into 3. The first one is the repeating factor of the two next lines.

Figure 7: Data Server Metadata



Again, the design being object-oriented, the layouts are polymorphic objects. Future types of layout may be introduced without any code change. The files also are polymorphic objects: support for new media or file storage systems will be done without difficulty. A layout can span several files, tapes or disks and a file can be shared amongst several layouts.

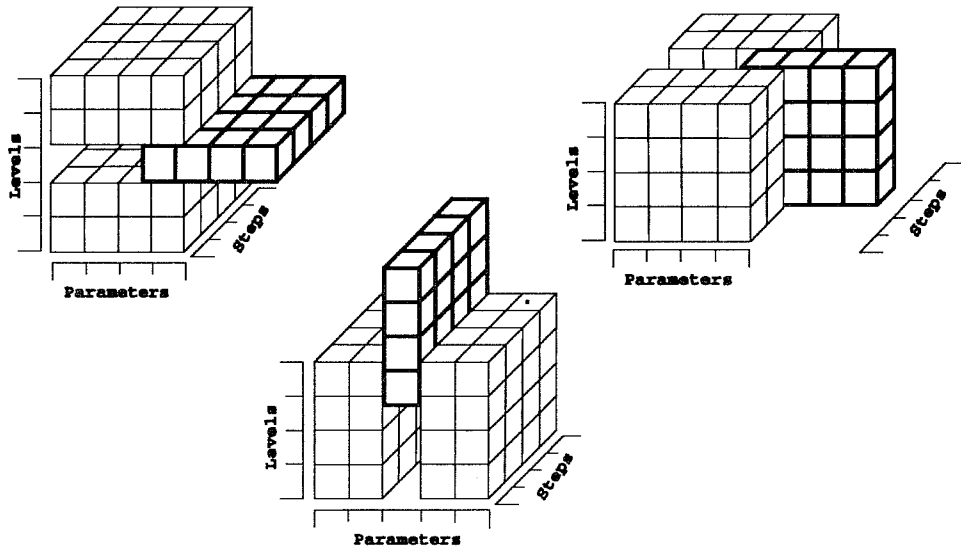
Each file knows how many layout lines point to them (the numbers shown in Figure 7). This is called reference counting and is used to perform garbage collection: files that have no pointer to them are automatically deleted.

3.6 Adding a new field to an existing archive object

The design described above gives us a new feature that did not exist in the previous system: incremental archiving. A field can be added later to an existing archive object. If one of its attributes has a new value, it is automatically inserted.

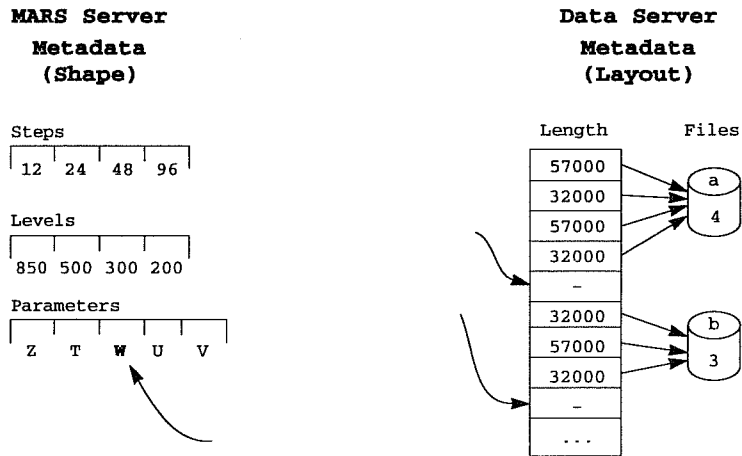
If an archive object has three dimensions, there are three ways it can grow, as shown in Figure 8:

Figure 8: Reshaping hypercubes



In general there are n ways to grow an n dimensional hypercube, by inserting an $n-1$ dimensional hyperplan. It is a lot easier to perform the same insertion on the metadata once it is split. If we archive a new parameter w in an archive object (Figure 9), we simply insert a string in an array (MARS Server) and some empty slots in another array (Data Server).

Figure 9: Updating metadata



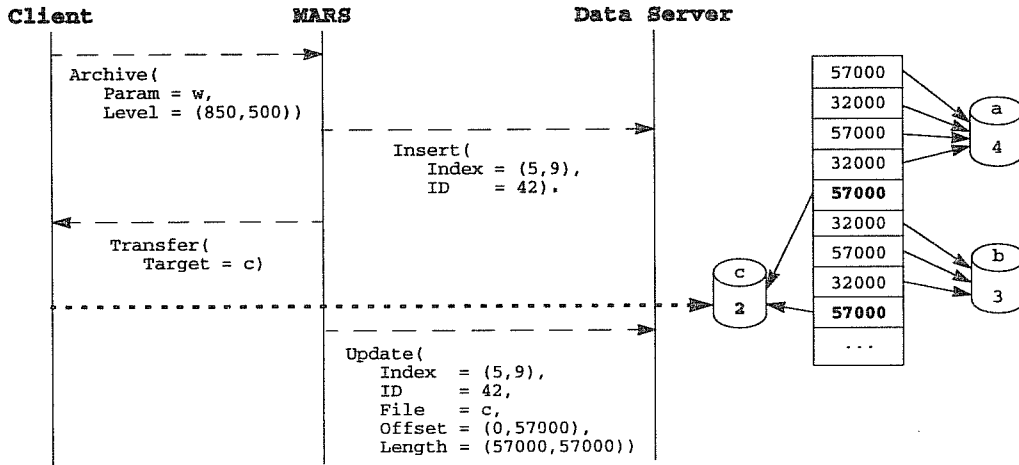
4. MARS in action

4.1 Archiving

When an archive request is issued from a client, the request descends the tree (Figure 6) until it reaches a shape. If it does not find a shape, the tree is grown accordingly, and a new empty shape is added. Figure 10 shows an incremental archive. The names at the top are the various components involved. Time flows from top to bottom. The dashed arrows are messages sent between the components. The dotted arrows are data transfers.

Files a and b contain data that has been archived earlier. Data is transferred from the client machine into a disk file c. This file contains two 57000 bytes fields. The layout is resized to accommodate the new fields, and it is updated to point to file c. The data held in c is still on disk, and said to be in “pre-archive” stage.

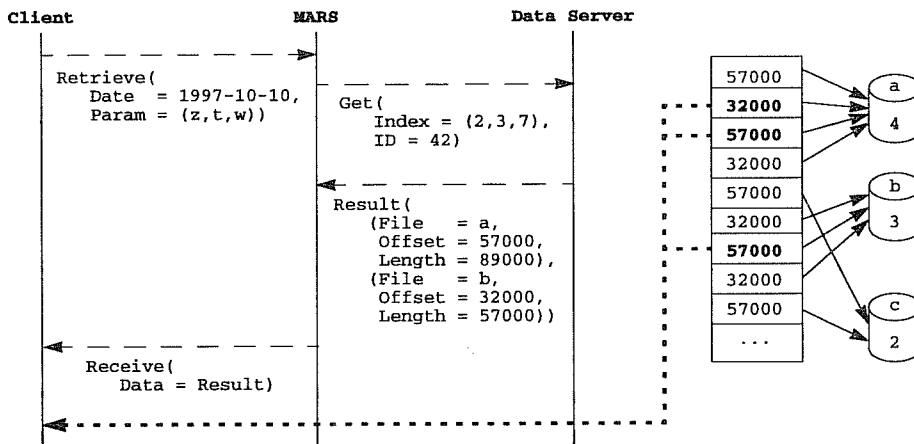
Figure 10: Incremental archiving



4.2 Retrieving from disk

While the data is still in “pre-archive” stage, retrieving is very fast. As in archiving, the MARS tree is visited in order to find one or more shapes matching the user request. The request is translated into a list of indexes in the layout, and data is read directly from disk files and sent to the user.

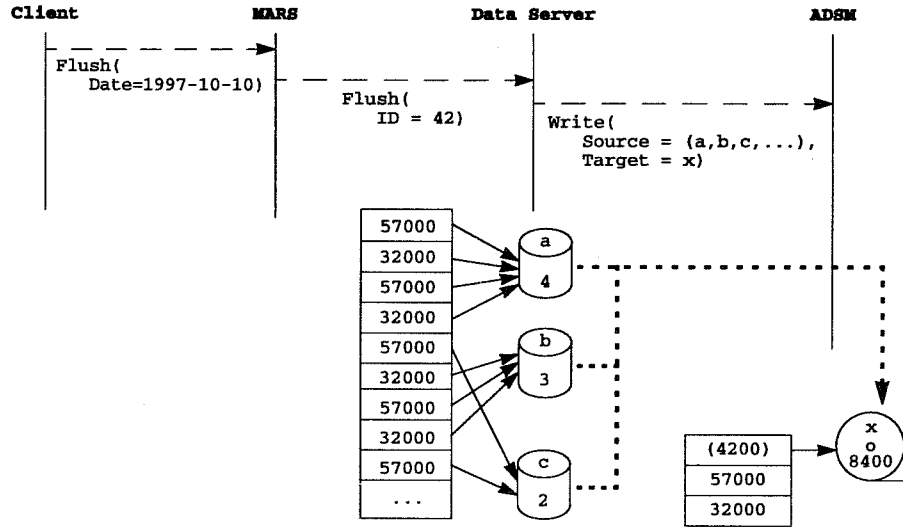
Figure 11: Retrieving, data is on disk



4.3 Flushing

Flushing occurs when an archive object is complete, or when the disks are full. A flush request writes all the fields that are in the “pre-archive” stage onto tape. The layout is updated to reflect the change, and the on-line data is deleted. The layout is then compacted. Figure 12 shows that layout 42 is composed of files a, b, c and more. These files are merged into a single tape file x. The layout is compacted from 8400 lines to three lines, noticing that the field sizes are 4200 times 57000 and 32000 bytes long. Once the tape file exists, the archive object is no longer in the “pre-archive” stage.

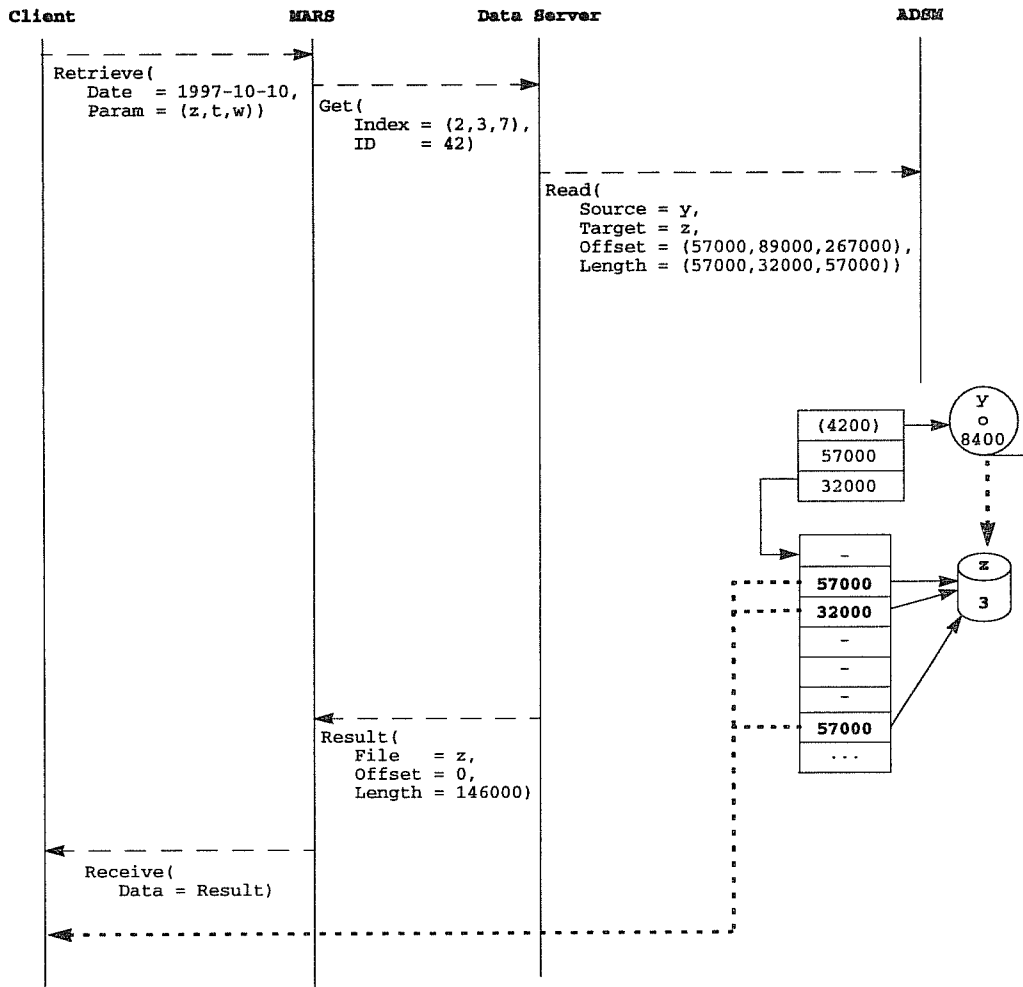
Figure 12: Flushing



4.4 Retrieving from tape

When the data is on tape, it must be first copied to a set of disks that are the MARS cache. ADSM is very useful as it can read portions of a tape file, so only the requested data is cached. When data is retrieved from a tape, a temporary layout, called the "cache layout" is created. This layout contains pointers to all the data that has been cached for a particular archive object. Figure 13 shows the flow of request and data. The requested data is copied from tape y into cache file z. The data is then sent to the user as described previously. The cache is emptied using a least-recently-used algorithm.

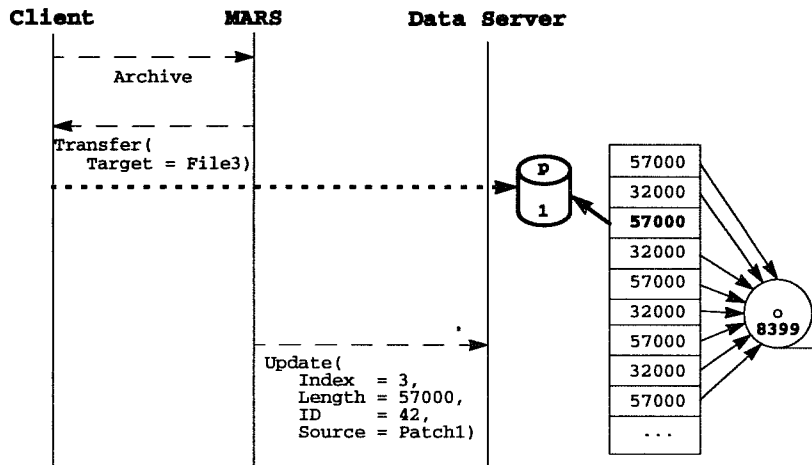
Figure 13: Retrieving data that is on tape



4.5 Patching

To correct a field that is wrong, the user simply needs to archive it again. If the layout has been flushed, it is mutated in its expanded form again, and the field is simply added as for a normal archive (Figure 14). The reference count of the tape file is decremented. If it reaches zero, that means that all the fields have been replaced, and the tape file is automatically deleted. The layout can be flushed again. In this case, only new data will be written to tape. The layout will now point to several tape files. When a layout is too fragmented (it points to too many tape files) it can be defragmented by copying all of its data back to disk and flushing it again, into a single tape file.

Figure 14: Patching



5. Conclusion

After one year of service, the new MARS lives up to its promise. It now contains 520 million meteorological fields, representing 32 Tbytes in only 180,000 files. More than 7 million requests have been processed. The metadata database represents 0.03% of the size of the archive.

This would not have been possible without an object-oriented design, the metadata being too complex for a relational approach, and yet very simple for an object-oriented approach. The system is extendable, new data types can be supported and the underlying storage management can evolve.