Simple query comparison between a relational database and an object-oriented database

Pierre Didelon[a,b] and Bertrand Morin[a,b]

[a]CEA/DSM/SAp, CE-Saclay, 91191 Gif/Yvette, France.
[b] IAP/TERAPIX, 98bis boulevard Arago, 75014 Paris, France.

ABSTRACT

We compare the efficiencies of a relational database, MySQL, and an object oriented database, Objectivity, for simple functionalities, like handling data produced by a reduction pipeline. For each test, we then challenge two designs as similar as possible, both capable to handle the same amount of useful data, and which both have the same functionalities implemented on top of the design. From this study, we then discuss which system is more suited or efficient for these types of applications. In particular, the impact of the design on the global application efficiency is addressed, as well as the advantages and limitations of the two systems. The tests show that relational system are more efficient for simple queries. The absence of a global query syntax for object oriented database turns out to be a disadvantage when dynamic behavior is needed or required. Moreover, it implies an additional effort in API (Application Programming Interface) development, which reduces the strength of object oriented system when handling complex data structure and related queries.

Keywords: Database, relational, object-oriented

1. INTRODUCTION

TERAPIX[1] is the data center in charge of handling and processing the data generated by the MegaCam camera[2,3] operated at CFHT. This camera, which will be the primary CFHT instrument for the next five years, is going to produce about 50 TB of data and will handle three big surveys which will generate more than 1 billion objects. In order to keep track of all the data and the processing history, a database has been designed to operate as a data repository *. In the early TERAPIX development phases, the database was thought to handle all the data down to a very fine granularity, like astronomical detections listed in catalog entry. Since only object oriented database management system would be able to operate at this level, for this purpose we first selected Objectivity. However, Objectivity turns out to have a low data retrieval speed, which considerably weakens the cross identification performances on catalogs stored in the DB. Furthermore, taking into account the poor support we have from Objectivity during the past two years, we therefore questioned the gains to choose an object oriented database (OODB hereafter). In fact, when practical reasons and the primary TERAPIX database queries are challenged against the need for handling 1 billion objects, it seems to TERAPIX more reasonable to downgrade the database specifications and to focus on a simple database design with limited processing functionalities. A more traditional DB technology based on a relational database (RDB hereafter), implemented as a free and open source distribution, like MySQL, was therefore preferred. In this approach the database operates at a higher level and is restricted to images, catalogs and global products handling, with almost no applications at the catalog entry level. Although this simplification permits relational technology to operate efficiently, some tests were planned to asses these assumptions and carefully compare the two technologies, at least for realistic use cases that will certainly occur in the TERAPIX context. †

Beyond, TERAPIX, the tests we describe in the following are also useful to compare the expected and foreseen behavior of the two technologies. Moreover the careful description of data used and the tests we carried out can be used as benchmarks for others products or technology.

Authors contact : P.Didelon : E-mail: pdidelon@cea.fr / B.Morin : E-mail: bmorin@cea.fr

*it is important to stress that TERAPIX is not in charge of archiving or distributing the final images and catalogues. So the TERAPIX database is only a local one for the processing history

†More information available at http://terapix.iap.fr/doc/OODBIX/DBcomp/index.htm

2. TESTS PRESENTATION

Our tests only focus on data extraction from the database, so any other considerations are not addressed here. For example, the efficiency of data loading in database, which was already tested under Objectivity, turned out to be satisfactory and was not longer felt as an issue in the new design and usage of the database under MySQL. Like indexing and updating data, these considerations might by the subjects of others comparison studies. Moreover further investigation of data retrieval or structuration, like cross-identification or re-clustering of data collection, are not addressed here, but these functionalities are build on top of the basic ones studied here.

Since relational technology limit, when handling complex data structures is imposed by joints between tables, while object technology is very efficient for that purpose through the explicit and direct links between objects, we first tested 1-n and n-n joints. The experiment were carried out on synthetic data as similar as possible, the main unavoidable difference being the links storage policy. An additional preliminary test has also been performed to compare the efficiency in scanning and retrieving one item from non indexed data. Efficiency of retrieving informations from indexed data have been obtained from results of the second test, mainly devoted to evaluate the crossing of 1-n joints.

Each serie of tests has then been performed two people working separately on two types of products representing two implementations of the MySQL and Objectivity technologies. It is widely admitted that both products are among the most efficient candidates of the two technologies, in terms of data transfer speed (For example see MySQL benchmarks [‡]). Because MySQL and Objectivity have different storing policies, it is necessary to define common concepts prior to design the tests. Data storage in RDB is organized in tables made of columns which define the content of the table, and rows were data are stored. The OODB equivalent to a RDB table is a list of objects of the same type (*class* in the Object terminology) stored together in the same bag without any other type/class of objects. Defined as such these concepts described the same storage organization and will be generically named *collections*. Within a collection each piece of information, object or row (*tuple* in relational terminology) will be denoted an *item*. Likewise, a numerical data value stored in the database will be named a *data attribute* or, more briefly, an *attribute*. Each attribute corresponds to a data member of an object or equivalently to the content of a table cell.

In order to compare the properties and performances of the two DB implementation options, we carried out at least three main tests. The tests conditions were similar and reflect a common use of similar data. For each result comparison, execution conditions of individual tests will be indicated. The 3 following test configurations have been defined and used :
- retrieval/accessing non-indexed data from a simple collection,
- crossing of 2 consecutive 1-n relations, from both sides, and
- crossing of one n-n relation.

2.1. Tests Conditions

For both technologies all tests have been performed on the same computer, namely an alpha EV5 with 1Gb memory running OSF1/Digital Unix V4.0D, and using the same disks, namely a SCSI device (7200 rpm). The I/O disk performances have been tested using the standard TERAPIX benchmark tool PerfIO 1.1, [§]. It gives a mean read speed of 13 Mb/s and a mean write speed of 4 Mb/s. Note that only the read speed has an impact on the results presented in this work.

The tests of the relational technology have been performed under MySQL version 3.22.22; those of the object oriented technology have been performed under Objectivity version 5.0.
In both cases, client, server and data have been installed and used on the same computer, therefore the network efficiency issue is irrelevant for the comparisons.

[‡] see http://www.mysql.com/information/benchmarks.html
[§] see http://terapix.iap.fr/hard/cluster/raid_config.html at bottom

Table 1. Time in seconds to retrieve one specified item from a non indexed collection.

Sample size	MySQL	Objectivity
10 000	0.084	0.53 (+/- 0.03)
40 000	0.35	1.8 (+/- 0.1)
70 000	0.57	3.9 (+/- 0.1)
100 000	0.81	5.4 (+/- 0.2
1 000 000	/	62. (+/- 2.)

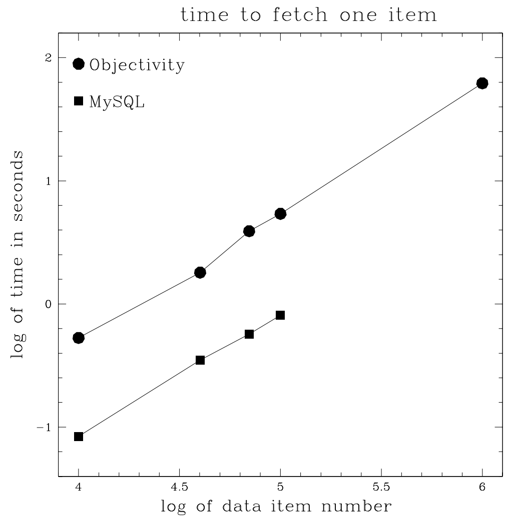

Figure 1. Comparison of times needed to access non index data as a function of sample size.

3. TESTS RESULTS

The times measured here are elapsed times, and each value is the mean of 4 to 10 measurements, depending of the convergence speed of each test. Therefore these times are representative of a "warm" data access mode. Since errors are small enough to be within the point sizes, they are never visible.

3.1. Access to non indexed data

In this test, the times to retrieve specified data without index usage are compared. Data are extracted either form a table, in the relational example, or from a container containing instances of one class type, in the object oriented example.

3.1.1. Sample size influence

We first illustrate the impact of the sample size (number of items) on the retrieval time.

The MySQL test is done using a table with 2 string attributes (varchar), each row being filled with 23 bytes. The Objectivity test is carried out using a container of objects with two string attributes of variable length, filled with about 23 characters, and one integer (32bit/4bytes). The collections are filled with synthetic data generated during the data creation. The data are then explored and selected by string matching.

During the test the whole sample is scanned, the retrieve condition is applied to each item of the collection, and the data contained in the matching items are load in memory. Conditions are defined to match only one element.

Discussion From the results given in Table 1 and Figure 1 we see that MySQL is 5 to 7 times faster than Objectivity. The time continuously increases with the sample size but the MySQL to Objectivity time ratio keeps constant over the whole range explored. This is primarily due to fact that object oriented access to data is performed at the object level and not at the attribute level. The object is first entirely loaded in memory, then access to the different object data attributes/parameters is performed.

3.1.2. Data size influence

Now, we try to measure the impact of the item size when the sample size (items number) is kept constant. So, this test illustrates also the trend with total size of the whole sample.

The MySQL test is done using a table with 2 attributes, one string containing a maximum of 20 characters and one text zone containing the required character number to match the size in each row. The Objectivity test is done using a container of objects with two string attributes of variable length, filled with the character numbers required to provide the size for each object, and one integer (32bit/4bytes). The data are explored and selected by string matching. For this comparison, only the beginning of the string containing a fix pattern and the item running number have been used. In both cases the sample size is equal to 10 000 items.

During MySQL tests computer was loaded at 30%, so a corrective factor of 0.7 must be applied to measured times. The final results express the time (in seconds) needed to Objectivity and MySQL to find one specified

Table 2. Time in seconds to retrieve one specified item from a non indexed collection as function of the total data size in Megabytes. Objectivity case.

Total Data Size	time
0.23	0.4
0.9	0.51
20	0.85
200	6.15
2 000	36.75

Table 3. Time in seconds to retrieve one specified item from a non indexed collection as function of the total data size in Megabytes. MySQL case.

Total Data Size	time
92	$1\times0.7 = 0.7$
300	$3\times0.7 = 2.1$
400	$4\times0.7 = 2.8$
600	$6\times0.7 = 4.2$

item among the whole sample and to fetch the data in memory. From the Tables 2, 3 and figure 2, one can see that, within the range where the comparison can be done, MySQL is about 4 times faster than Objectivity.

3.2. crossing of two consecutive 1-n joints

Database context and test motivation The time needed to navigate between collections is compared in this test. The relation followed to navigate between 2 collections can be of two types :
one item of a collection is linked to at most one object of the other collection. This is a so-called “to 1” relation.
one item of a collection can be linked to n objects of the other collection. This corresponds to a “to n” relation.

Moreover since link can be followed in one direction or the other, it results in three possible types of relation :

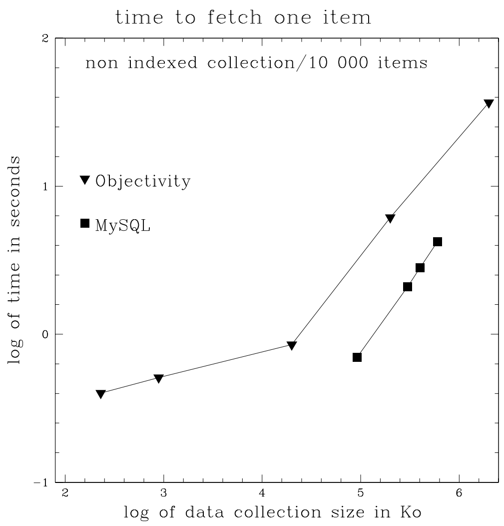

Figure 2. Comparison of time needed to access non index data as a function of data size.

- 1-1 : 1 to 1.
- 1-n : 1 to n, and n to 1 on the other way
- n-n : n to n.

In object oriented technology the links are explicitly maintained individually by each items (objects), so it is easy to follow them and this process must be fast. In relational mode the links are only implicit between items (rows/tuples) of different tables. They need to be established each time a relation is followed. This is a straightforward and fast task for the "to 1" case, but not so easy for "to n" relation. In this case, the link construction mechanism (Cartesian product of the two concerned tables) can be heavily time consuming, in particular when big tables are handled (lot of items/rows/tuples, big rows containing a lot of attributes or both). This well-known limitation in relational technology is often referred to as the "relational wall" by object oriented database developers. To quantify this limitation, in a first step we used 1-n relations and performed navigation tests in both the "to 1" and the "to n" ways. To increase the impact of the joint we consider here two consecutive 1-n joints. Our implementation is rather specific in order to match a realistic situation which can occur at TERAPIX or any data center concerned by handling the data processing history.

Implementation We consider three collections linked by two 1-n relations. The first collection contains X Images, the second collection contains X/10 Processes and the third contains X/100 Configuration files. Hence, these three collections are linked by two 1-n relations where n is equal to 10.

On one way, each process uses 1 configuration and each image is produced by 1 process, so only 1 configuration is used per image creation. On the reverse way, each configuration have been used by 10 processes, and each process have generated 10 images, so 1 configuration has been used to generate 100 images.

In order to measure the time needed to follow two consecutive 1-n relations in both directions, we carried out two different tests :

1. a search process to identify which configuration file has been used to create an image. This is the "to 1" direction,
2. a search process to find all images which have been created using a given configuration file. This is the "to n" direction.

The MySQL test is done using tables with either two string attributes (Images and Process tables in which link must be stored), or one string attribute (Configuration files in which no link has to be stored). In the second case, one attribute is used as an identifier, the other one is used, when needed, as a key to establish the link with items of the other table. During MySQL tests computer was loaded at 20%, so a corrective factor of 0.8 must be applied to measured times.

The Objectivity test used containers of objects. One string attribute is used as an identifier. The links to items of the others collections are explicitly stored in the object, and are directly followed in order to retrieved the required information.

The collections are filled with synthetic data generated during their creation. Links are established using the *random()* function of the C library. This choice prevents undesirable ordering which may perturb the study. The data are explored and selected by string matching process. An index based on identifier attribute is established and is used to speed up the data access.

The comparison tests were carried out for different sample sizes, using from 10 000 to 100 000 images. No output, on screen or in a file, is requested during the tests, so the results really express the times one needed to find one specified item among the whole sample and to fetch attribute data in memory.

3.2.1. on the "to 1" way

For both MySQL or Objectivity, the time to retrieve one configuration file from one image identifier does not depend of the sample size:

- MySQL : 0.0012 +/- 0.0005 sec. ($= 0.0015 \times 0.8$)
- Objectivity : 0.20 +/- 0.01 sec.

In fact, in the case of the n to 1 direction, the time to cross the two relations is so short that the test mainly measures the time needed to access, via index, to the required Image from its identifier. Therefore, for both technologies, it illustrates the time needed to access to indexed data.

Note that the indexed data access is about 150 times faster under MySQL than under Objectivity. This illustrates the fact that the initialization process is much complicated and time consuming in Object oriented database than than in relational databases. Nevertheless, even if the initialization time (0.14 sec for Objectivity) is subtracted from the test time, Objectivity is still 50 times slower than MySQL. This significant difference is indeed hard to understand. Two effects mix together:

- the access to data through index and
- the navigation through a "to 1" relation.

A preliminary specific test devoted to measure the time needed to access indexed data would distangle these two effects and perhaps clarify the comparison. However, as it will be demonstrated later, the navigation process is very efficient in OODB and cannot account for such a difference. Is is likely that the main impact is due to the low data access efficiency, even through the index process. This is confirmed by measurements and results of tests performed on independent data, presented in Ref. 4. Additional tests with a subsequent version of Objectivity must be performed to check any improvement with respect to our quite old version of Objectivity we used here.

3.2.2. on the "to n" way

The time to retrieve 100 images from one configuration file identifier is given in Table 4.

Table 4 and figure 3 clearly show the limitation of relational database when the crossing 1-n relation is carried out in the less favorable way (to n). In that context MySQL is slower than Objectivity, and its limitation rapidly increases with sample size.

Table 4. Times, in seconds, needed to retrieve 100 images from one configuration file identifier and to fetch data in memory, for Objectivity and MySQL.

Sample size	MySQL	Objectivity
10 000	$0.89 \times 0.8 = 0.71$	0.47
40 000	$4.07 \times 0.8 = 3.26$	0.82
70 000	$7.94 \times 0.8 = 6.35$	0.90
100 000	$11.95 \times 0.8 = 9.56$	0.98
1 000 000	/	1.36

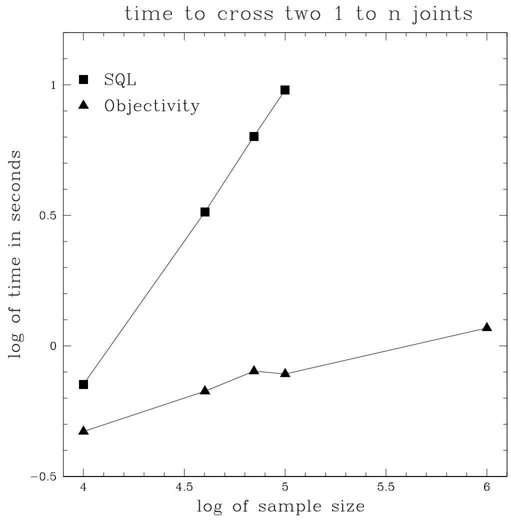

Figure 3. Comparison of times needed to retrieve 100 items across two consecutive 1 to n joint

Can the joint process with SQL be avoided? Although the previous tests clearly demonstrate the strength of OODBs, it is important to keep in mind two key points which have to be balanced prior to reject RDB:

- the data retrieval from one indexed table is very fast in RDB via SQL,
- in OODB, object to object navigation is not made via an ad hoc language, like OQL would be, but in an API, and so needs code development.

It is indeed possible to do the same job for RDB than for OODB, provided one write a code to construct an API which will perform the joints process instead of SQL. If so, direct selection in one table can be made iteratively on all relevant tables. In the present case one can imagine to perform the following steps :

- step_1 : select 1 Config,
- step_2 : select 10 Process,
- step_3 : select 100 Images.

This scheme procudes 111 selections on indexed data. Since the individual time of such a functionality is 0.0012 for MySQL, the total time will then be 0.133 seconds. Even if we take into account that the two first steps do return 10 items, multiplying these two steps by 10 (a conservative overestimation) would result in 210 select, that is a total time of 0.252 seconds. This solution allows the DB users to built up a process which spend a comparable time to object oriented technology joint process. Furthermore, it is also no longer sensitive to sample size, at least in the domain where our tests were carried out. If the time to retrieve data from a key

Table 5. The times, in seconds, needed to retrieve 50 images from one imageList or 5 imageList from one image, crossing a n-n joint, for Objectivity and MySQL.

	get 5 imageList from 1 image		get 50 images from 1 imageList	
Sample size	MySQL	Objectivity	MySQL	Objectivity
10 000	$1.4 \times 0.7 = 0.98$	0.14	$1 \times 0.7 = 0.7$	0.15
20 000	$3.3 \times 0.7 = 2.31$	/	$2.7 \times 0.7 = 1.89$	/
30 000	$5.2 \times 0.7 = 3.64$	/	$4.3 \times 0.7 = 3.01$	/
40 000	$7.1 \times 0.7 = 4.97$	0.14	$6.2 \times 0.7 = 4.34$	0.14
70 000	/	0.14	/	0.15
100 000	/	0.14	/	0.15

critically depends (above the linear level) on the amount of data returned and if the 1-to-n relation can lead to large amount of items, the impact of the relation cardinality n has to be tested for both the relational and object oriented cases, for different values of n, to be exhaustive.

This crude estimation seems to show that investing the same manpower in database API for both case, would result in a deep improvement of relational database performance when crossing joints.

3.3. crossing one n-n joint

In the previous section we have shown that following 1-n relations is more efficient with OODB that RDB. Now, we compare the performances of the 2 technologies regarding the navigation through n-n relations. A n-n relation denotes 2 collections, each element of 1 collection is linked to n elements of the other collection. While this task is easily handled by OODB, as for the 1-n relation discussed above, with direct explicit links stored as object pointers, this is more complex fro RDB. They need to introduce an intermediate table, in order to handle the cross references between the 2 collections/tables, and to store there a row identifier which can be used as a key between tables. So, while OODBs follow 1 list of links, RDBs need to cross-correlate 3 tables through 2 joints, that is 2 consecutive Cartesian products. This is the worst situation for RDB, and we can anticipate for this case a very disadvantageous result.

Implementation An interesting practical case of such a pattern in the TERAPIX context would be a mix of Images (Im hereafter) and List of Images (LoIm hereafter). Obviously 1 LoIm contains several Im and, conversely, 1 Im can be used in several LoIm, build for different purposes .

We consider here that each LoIm contains 50 Im, and that each Im can be listed in 5 LoIm. So, if the sample contains X Im, it must contains X/10 LoIm.

MySQL is using 2 tables with one string attribute, to store respectively the Im and LoIm identifiers, and 1 table with 2 string attributes, to store the couples of Im and LoIm identifiers which handle the links between the 2 other tables. During MySQL tests, computer was loaded at 30%, so a corrective factor of 0.7 must be applied to measured times.

The Objectivity test uses 2 containers of objects with one string attribute being used as an identifier. The links to items of the others collections are explicitly stored in the object, and these links can be followed directly to retrieved the required information.

The items of the collection are filled with synthetic data generated during their creation process, and contain a basic string (Image_ or ImageList_) followed by the running item number. Links are established using the *random()* function to prevent undesirable ordering. The data are explored and selected by string matching. Indexes based on identifier attributes are established and used to speed up the data access.

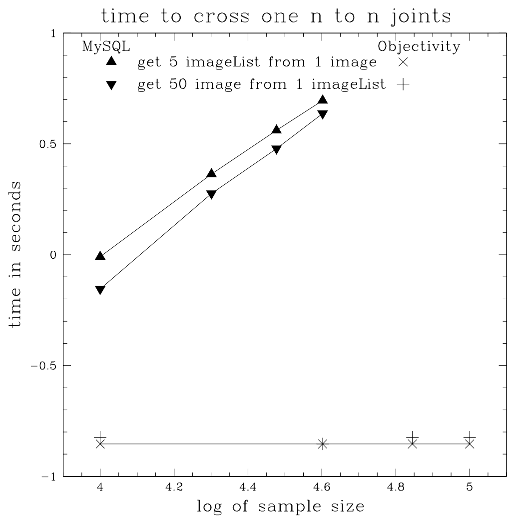

Figure 4. Comparison of times needed to cross a n-n joint in both direction.

The n-n joint have been followed in both directions. Starting with 1 image it gives access to 5 List of Images, while starting with 1 List of Image it gives access to 50 Images. The comparison tests were performed for different sample sizes, using from 10 000 to 100 000 images.

Discussion Table 5 and figure 4 show unambiguously the limitation of relational database when crossing a n-n relation. In that context, MySQL is much slower than Objectivity, with a rapidly increasing difference with sample size.

Like for 1-n relation, we can evaluate roughly what would be the time need to perform the navigation in the API in order to prevent the Cartesian product tables made by SQL. For the worst case we will have first to select one LoIm in the appropriate table. Then, we must select all Im identifiers corresponding to this LoIm in the cross correlation table. This will return 50 identifiers. Finally we have to select the 50 Im in the appropriate table. So, it results in 51 selections, or to be more conservative as we were previously, at most 101 selections on indexed data. Therefore the time would be 0.06 to 0.12 second. Times are then comparable to object oriented technology and is again also no longer influenced by sample size.

Once again, this crude estimation seems to show that implementing the navigation through joints in API would strongly improve the RDB performances when crossing joints, rising them to those of OODB.

4. CONCLUSION

When using simple queries which apply to only one collection of data (table in relational case), RDB is more efficient than OODB, regardless the data access (index or any other ways). The poor behavior observed with Objectivity regarding indexed access to data is surprising. We do not yet fully understand this, and more investigation must be performed with Objectivity.

For more complex queries where several collections are concerned, OODB is much more efficient and scalable. Nevertheless the absence of a global query syntax for object oriented database is a disadvantage. OQL is not really a standard as compared to the general use of SQL for relational databases. Consequently putting efforts in the API development for an RDB, as compared to similar investment for an OODB, seems a good way to go. Our rough estimate shows that functionalities and optimization can be implemented, in both cases, at the application level, it will certainly allow RDB to be as efficient as OODB.

The results shown is this paper are only valid for our specific granularity. It corresponds to a strategy were products (images,catalogs...) are handled as unbreakable entities. To handle smaller granularity, like for example individual catalog entries as well as the links between them (like cross-identifications), then it is not obvious that RDB can be still competitive with OODB. Link is in principle better handled directly by OODB, while RDB needs additional breaking down effort. For big quantity of complex data, OODB is certainly still advantageous. For the same reasons, handling dynamic design will be easier and perhaps more efficient with OODB than RDB.
Nevertheless these assumptions would need additional specific tests.

ACKNOWLEDGMENTS

We deeply thank the Terapix team for the fruitful discussions that took place during this work and the carefull reading of the manuscript.
This work is partly funded by the European RTD contract HPRI-CT-2001-50029 "AstroWise"

REFERENCES

1. Mellier, Y., Bertin, E., Bonnarel, F., de Coussemaker, N., Dantel-Fort, M., Didelon, P., Domisse, P., l'Hermitte, J., Magnard, F., McCracken, H., Missonnier, G., Morin, B., Radovich, M., Tissier, G. . "TERAPIX Progress Report 2002". http://terapix.iap.fr/doc/doc.html
2. O. Boulade et al., "Development of MegaCam, the next generation wide-field imaging camera for the 3.6m Canada-France-Hawaii Telescope," in *Optical and IR Telescope Instrumentation and Detectors*, M. Iye and A. F. Moorwood, eds., *Proc. SPIE* **4008**, pp. 657–668, 2000.
3. O. Boulade et al., "MegaCam: the new Canada-France-Hawaii Telescope wide-field imaging camera," in *Instrument Design and Performance for Optical/Infrared Ground Based Telescopes*, M. Iye and A. F. Moorwood, eds., *Proc. SPIE* **4841**, 2002.
4. P. Didelon, "Dynamic design impact on data retrieval speed," in *Virtual Observatories*, A. Szalay, ed., *Proc. SPIE* **4846**, this meeting, 2002