MonetDB/SQL Meets SkyServer: the Challenges of a Scientific Database

M. Ivanova N. Nes R. Goncalves M. Kersten

Centrum voor Wiskunde en Informatica
Kruislaan 413, Amsterdam
{*M.Ivanova, Niels.Nes, R.A.Goncalves, Martin.Kersten*}*@cwi.nl*

Abstract

This paper presents our experiences in porting the Sloan Digital Sky Survey(SDSS)/ SkyServer to the state-of-the-art open source database system MonetDB/SQL. SDSS acts as a well-documented benchmark for scientific database management. We have achieved a fully functional prototype for the personal SkyServer, to be downloaded from our site. The lessons learned are 1) the column store approach of MonetDB demonstrates a great potential in the world of scientific databases. However, the application also challenged the functionality of our implementation and revealed that a fully operational SQL environment is needed, e.g. including persistent stored modules; 2) the initial performance is competitive to the reference platform, MS SQL Server 2005, and 3) the analysis of SDSS query traces hints at several techniques to boost performance by utilizing repetitive behavior and zoom-in/zoom-out access patterns, that are currently not captured by the system.

1 Introduction

In the recent Lowell report [1] the scientific world has been identified as a major challenge for database technology. Current solutions are far from perfect and rethinking in several dimensions is urgently needed, e.g. support for ARRAY- based processing in the context of SQL [17].

A scientific database can be characterized as a large data-warehouse, where ad-hoc querying is prevalent. However, unlike its administrative business counterpart, the data is much more noisy, requires extensive numeric processing, and calls for fast extract/transform/load operations in the scientific work flow. The scientific archive we are working with is the Sloan Digital Sky Survey(SDSS)[14], an astronomy survey aiming at creating a digital map of a big part of the Universe. The survey has already collected several terabytes of data.

Crossing the great divide between database and astronomy research has been pioneered by J. Gray in the SkyServer project[6], which provides public access to the SDSS warehouse for astronomers and a wide public. After a decade of disillusions working with emerging persistent object-oriented languages (C++), the SkyServer team convincingly showed the potentials of a SQL-based solution, challenging the SQL server team to support the complexity of a real-life astronomy application.

Despite the wide availability of the data and application logic, few database groups have made an attempt to port the application to their DBMS of choice. There have been informal reports on attempts to get it working on Oracle and DB2, but none has reached a level of maturity to compare with and learn from. Often the SQL-dialect is not supported, or calls for missing features in the kernel or optimizer. Likewise, financial constraints may hinder setting up a multi terabyte database in a research lab.

The MonetDB/SkyServer project has been set up as an experimental study of large scale databases based on the MonetDB platform [9], an open-source database system developed for over a decade at CWI. The goal of the project is to provide a counter proof for the MS SQL Server implementation and to act as a platform for developing novel algorithms to support the data management challenges in the domain of scientific databases. In this paper we address the following issues: 1) functional improvements in MonetDB and adaptations of the original SkyServer application to get it up and running, 2) the initial performance figures, and 3) the analysis of query traces that gives insights about the system usage and shows opportunities for performance improvements.

The SkyServer schema was designed around the MS SQL Server engine and application load. Auxiliary tables and covering indices are heavily used to speed up important query classes and even to compensate for limitations encountered in the MS SQL Server query opti-

19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)
0-7695-2868-6/07 $25.00 © 2007 IEEE

mizer. A large collection of application specific functions are defined as SQL-functions.

We aimed at porting SkyServer as closely as possible and also at performing all experiments on the same hardware platform. This required significant investment in developing features missing in MonetDB/SQL. Shortly after our first porting attempt, it became clear that a 10% space reduction is easily attainable without significant performance loss, due to the column store approach in MonetDB.

In parallel, we extensively studied the query logs of SkyServer from August 2006. This sequence of 1.3M queries was analyzed for repetitive patterns and hints for optimization. Aside from the inherent functional structure of the website [15], it elicits a user behavior amenable to multi-query optimization.

The experience in SkyServer porting hints at a number of research topics calling for our attention. Multi-query optimization in a scientific field is less complex than foreseen. The zoom-in/zoom-out behavior and querying focusing on the same area can easily be recognized and exploited for pre-caching. Usage statistics reveal that some pieces of data are much more interesting for the scientists than others. That calls for investigating indexing techniques differentiating data depending on the usage frequency. Since SQL functions, including table-valued ones, are intensively used, their optimization becomes important for the system performance. Some of the possible approaches are caching and re-using results, and function in-lining.

One of the main purposes of SkyServer is to support astronomers in their research activity. The latter includes a variety of data mining tasks, such as finding interesting objects, correlations, and classifications. Besides retrieving the stored data, data mining algorithms require high performance, an area where MonetDB is a promising candidate.

The rest of the paper is organized as follows: the next section presents the main distinguishing properties of MonetDB/SQL. Section 3 describes our experiences in porting the SkyServer application, and Section 4 presents our observations from the query log analysis. The evaluation of the SkyServer port in terms of functionality and performance is presented in Section 5. We discuss directions for future work in Section 6 and conclude in Section 7.

2 MonetDB/SQL

In this section we briefly introduce the MonetDB server and SQL compiler. A growing class of database engines are geared at exploitation of a column-oriented store [16, 2]. In this field, relational tables are broken vertically, with each column representing a single relational attribute. This approach leads to a much simplified system architecture[3] and opens routes to increase performance through compression[18]. The benefits come from a better streamlining of the data flow from disk through memory into the CPU caches. Column-oriented data stores are particularly beneficial in data warehousing and data mining applications, which are often used on scientific databases. The primary reason is that most applications do not need the hundreds of columns of a relational table with scientific measurement data, but merely require looking at just a few at a time. The immediate benefit of the column store approach is that only data relevant for processing is fetched from disk.

MonetDB is a fully functional column store developed over a decade at CWI. It consists of a two-layered architecture of a database server and a number of front-ends. The server is addressed in a proprietary language, called MonetDB Assembly Language(MAL). MAL is a relational algebra language derived from MIL[3]. It supports a large collection of relational primitives, functions, and easy linkage with user defined functions. The operators work on the basis that each produces a materialized result. Moreover, the operators encode runtime optimization decisions, which in other systems are part of a cost-based optimizer. For example, the MAL join operator makes a runtime decision about utilization of additional indices, exploitation of sort-orders, and data type specific opportunities. It results in encapsulation of several hundreds of highly tuned join algorithms.

This approach significantly simplifies the front-end compilers. The SQL front-end parses SQL queries and compiles them into MAL plans exploiting the SQL language and schema semantics. It should (and can) only focus on the volume reductions achievable. The front-end compiler also selects MAL optimizer components to be activated, e.g. common expression elimination, dead code removal, parallelism, etc.

In this way a three-tier optimizer architecture is achieved with a clear division of tasks. The bottom layer focuses on operational optimization using the actual state of the machine. The top layer is geared at exploiting the schema semantics, and the middle layer is geared at tactical decisions. It is the place to decide on e.g. (pre-)caching results, scheduling, etc. For more information we refer to the MonetDB Version 5 documentation at http://monetdb.cwi.nl/.

3 Porting experiences

Data management of SkyServer is provided by MS SQL Server. Next, we describe our experiences in port-

19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)
0-7695-2868-6/07 $25.00 © 2007 IEEE

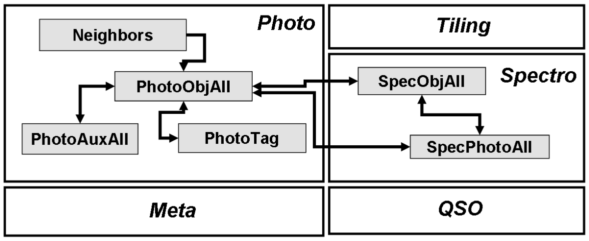

Figure 1. SDSS Database Schema

ing MySkyServer to the MonetDB platform. MySkyServer is a sub-project of SkyServer providing a 1% subset of its data for personal download and experimentation. The port and its evaluation were done on the currently available MyBestDR4 data set[1].

Porting a sizable application to a new database platform, especially a research vehicle, is a major undertaking. Unlike its commercial counterparts, bits and pieces of the open-source version have not been developed to the fullest extent, let alone tested in all areas. Hence, SkyServer porting gave a strong impulse in developing the missing MonetDB features and also served as a serious platform for evaluating system functionality and performance.

3.1 SkyServer schema

The SkyServer schema is a 200-page document with 79 tables, 40 views, and 156 functions of which 29 are table valued functions. By and large, any team involved in porting the application is faced with a substantial effort to get it to run.

The schema, shown in Figure 1, is organized in five sections among which PHOTO and SPECTRO contain the most important factual data from the SDSS survey. The PHOTO section has a star-flake structure centered in the PHOTOOBJALL table and its companion PHOTOAUXALL. This table contains 446 columns, which already stresses the capabilities of most (commercial) DBMSs. A single record occupies already 1824 bytes, and most of the fields are real numbers representing CCD measurements. The table contains more than 270 million rows. The main table of the SPECTRO part of the schema contains the spectrum measurements for approximately 1% of the photo data.

In order to handle this demanding application, the MS SQL Server implementation uses a variety of indexing techniques and auxiliary tables. All tables have primary and foreign key constraints supported by B-tree indices. In addition, many tables have covering indices on one or more columns, speeding up searches if any of the columns are included in a WHERE clause. Several auxiliary tables are used to speed up important classes of queries. For example, the NEIGHBORS table groups together pairs of SDSS objects within an a priori given distance bound of 0.5 arcminutes. This results in approximately 9 closest neighbors precomputed and stored per object. The PHOTOTAG table is a vertical partition of the PHOTOOBJALL table that stores redundantly its most popular 100+ columns. This redundancy aims to alleviate the record-based access problem, i.e. that the entire record of hundreds of columns needs to be read from disk even if just a few columns are needed. Often, queries select correlated columns from the photo- and spectrum data tables. Those queries are sped up by a pre-computed join stored in the SPECPHOTOALL table. To help the optimizer in some situations that are hard for automatic detection, auxiliary tables are introduced to lead the selection on the main tables.

3.2 Schema adaptation

The porting started with a schema adaptation which required MS SQL Server specifics in the schema definition to be cast to the standard SQL:2003 which MonetDB aims to adhere to. Examples include data types (”datetime” corresponds to ”timestamp” in the standard), SQL extensions specifying hints for data storage (PRIMARY KEY CLUSTERED), etc. Some identifiers chosen in SkyServer schema clashed with SQL:2003 reserved words, such as ”dec”, ”match”, and ”row”.

The conversion of the programmable part of the schema from MS SQL Server syntax into a SQL:2003 syntax was time consuming and required some manual work. This included tasks from simple replacement of string concatenation (’+’ vs. the standard ’||’) to implementation of MS specific operations and functions, such as bit-wise logical AND and OR and the STUFF function.

Besides the formal syntax changes, the database schema was logically modified to some extent. Utilizing our observation about data redundancy and the advantages of column based storage of MonetDB, we replaced some of the tables replicating data for performance purposes with views. We evaluated the benefits of this replacement by investigating its effect on the storage requirements and query performance. As a result PHOTOTAG and SPECPHOTOALL tables were dropped from the schema saving approximately 10% of the storage needs. However, the NEIGHBORS table based on computed distances between objects proved to be much more efficient than the computed view and we kept it as it is. An another modification is that we limited the index support to primary and foreign keys, and

[1] Meanwhile, a new data set MyBestDR5 was released.

19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)
0-7695-2868-6/07 $25.00 © 2007 IEEE

did not create the covering non clustered indices, introduces in the original schema for performance purposes.

3.3 Adaptation of functionality

We implemented in MonetDB/SQL the SkyServer functions related to the query capabilities, leaving web site related functionality for the future. A major change with respect to the original schema is the implementation of spatial access methods. While SkyServer supports three methods: HTM, Zones, and Regions[5], we implemented only the Zones algorithm, a choice justified by results reported by the SkyServer team[4]. In addition to being easily implementable entirely in SQL, the method brings benefits like avoiding the need for a foreign HTM library, and enabling the use of an SQL optimizer. This choice provided successfully spatial access functionality needed by the majority of the query log, leaving aside only a small percentage of queries that call some of the methods in the HTM library directly.

Porting SkyServer exposed as major deficiency in MonetDB the lack of Persistent Stored Modules (PSM) functionality. Our intention to stay as closely as possible to the standard drove us to implement this functionality in the SQL compiler. The clean separation of the SQL compiler and MonetDB server proved pivotal in reaching a functional working version within just two months. The lack of SQL bindings to C-functions was equally easily resolved. In order to provide a platform for comparable experiments we ported the enhanced MonetDB version onto the same Windows platform where the SkyServer distribution was running.

4 Query log analysis

In this section we present our findings over the 1.3M query log. The main purpose of the log analysis was to obtain properties and common patterns of usage that suggest hints for increasing the performance. The analysis was performed over a trace of 1.2M queries obtained by cleaning out the erroneous queries, and those rejected by the server due to violation of execution time or query cardinality limitations.

4.1 Spatial properties

The first observation is that spatial properties of the data are intensively used in the queries: naturally the celestial objects are identified and extracted based on their position in the sky defined typically in the equatorial coordinate system as a pair of right ascension and declination. As a result 83% of the queries contain calls to some of the functions providing spatial access, such as FGETNEARBYOBJEQ, FGETOBJFROMRECT, etc. Hence, it is very important that the system provides an efficient implementation of the spatial access functions.

4.2 Hot data sets and common patterns

Even though the SkyServer schema contains more than 70 tables, a relatively small core of photo and spectrum tables is accessed in the queries. Furthermore, the queries on those tables follow a limited number of patterns [11].

We also observed a correlation between entry points in the SkyServer web site interface and the query patterns in the SQL cache. One of the most common query patterns responsible for approximately 25% of the queries, is directly generated by the interface form for radial search, while several others are its variants. The query, illustrated below, selects the most popular properties of primary photo objects in a given sky location.

```
SELECT p.objID, p.run, p.rerun,
   p.camcol, p.field, p.obj, p.type, ...
FROM fGetNearbyObjEq(195,2.5,3) n,
   PhotoPrimary p
WHERE n.objID=p.objID
LIMIT 10;
```

The concentration of query activity around a small number of tables and of a limited number of patterns, hints at a workload-based optimizer infrastructure. Instead of providing a general optimizer with thousands of rules covering all possible queries, the optimizer in such a scientific application can be tuned towards the workload. This idea aligns with the MonetDB's optimizer toolkit approach, where specific optimizer modules can be activated by the front-end compilers. This way, the optimizer avoids exploring large portions of the space of possible query plans.

All areas of the SDSS footprint are visited in a month, but not all sky locations are equally popular, as illustrated in Figure 2. The most often accessed area, with more than 10K queries per month, corresponds to the default values in the web form for radial search. Evidently, interesting places described on the web site also have a high probability of access with more than 100 queries per month. Caching and re-usage of such results is feasible and can be learned without DBA interaction.

The divergence in the access frequency of spatial areas hints at an opportunity to speed up the spatial access by utilizing indexing methods providing fast access to popular data, such as crackers [7].

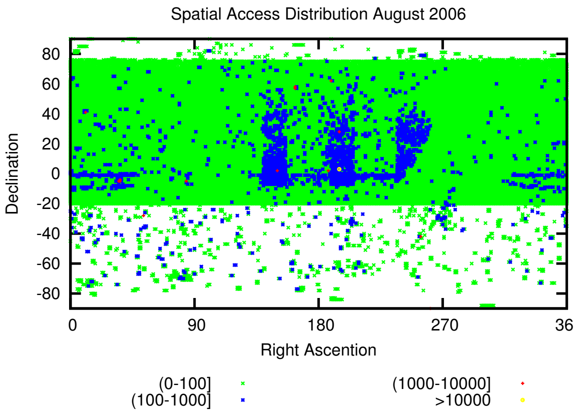

Figure 2. Query coverage on the sky

4.3 Inexperienced usage

Inexperienced user queries can be a real danger for a scientific database. We observed a statistically significant number of queries prematurely terminated by the server (due to running against the CPU time limit) reflecting inexperienced behavior. For example, due to the lack of a join condition, many queries compute Cartesian product of the 200M-row PHOTOOBJALL table with itself or with some of its correlated tables, such as the 7G-row PHOTOPROFILE table. Some queries contain predicates with doubtful use, such as comparing the mean flux (a real number) of an object to be equal to an integer value. We also encountered function calls with badly chosen parameters, e.g. negative right ascension, or coordinates in the Southern hemisphere that are not in the footprint of the SDSS survey. A number of queries with low selectivity have very large cardinality and generate a lot of network traffic. While this may be the real intention in some cases, in others it is probably due to mistakes. The above observations show that the time and cardinality limits put on query execution in SkyServer are indeed helpful.

4.4 Manual and script-based interaction

We observed two types of interactions to the server: manual and script-based. The manual traces reflect user interaction with relatively long inter-query intervals. Such a query sequence is often gradually refined in terms of more selective predicate values or adjustment of spatial parameters that takes the form of *zoom-in/zoom-out* behavior.

Script-based behavior is characterized with very short inter-query intervals and a small standard deviation. The script-based queries constitute more than 84% of the trace log when we set the threshold of those intervals to 15 seconds. They come in two flavors: sky scanners, traversing big areas of the sky, and point tracers, focusing on the same area. Depending on the logic in the script the system can predict future queries. Such predictions can be used to steer caching and re-using of query results.

4.5 Spatial overlap

We observed that many queries have some kind of spatial overlap relationship such as equality, containment, or overlap of the accessed sky areas. The analysis of 1M queries with valid parameters of their spatial access functions showed that 24% of them participate in some overlapping sequence where the sequences were considered on a day by day basis. The sequence length varies from 2 to 6246 steps with a mean value of 9.4 elements.

We considered five types of relationships between two subsequently queried spatial areas:

- equality '='
- containment in previous or zoom-in 'I'
- containing the previous or zoom-out 'O'.
- a version of zoom-out that is contained in earlier queries 'o'
- overlap 'x'

From a query execution point of view, the equality and zoom-in patterns mean that spatial search of the current query can be completely answered using the results from the previous query spatial search. Zoom out and overlap patterns allow for reusing the previous result set for partially answering of the current search but also require a new search for the uncovered part of the spatial area.

Overlapping patterns like '=Io=I' and '=IoIo===o' are typical for traces of manual interaction. Long equality patterns like '======' characterize query sequences generated by script-based interaction.

The detailed pattern statistics obtained over 1M spatial queries is shown in Table 4.5 together with the distribution for pairs of patterns. The prevailing pattern is equality (71%), followed by overlap (18%). The remaining 11 % describe zoom-in/zoom-out behaviour. The big percentage of an equality relationship among subsequent queries suggests a multi-query optimizer that reuses the result of spatial search functions.

19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)
0-7695-2868-6/07 $25.00 © 2007 IEEE

Table 1. Frequency of overlapping patterns

Pat	Total	Following symbol			
		=	I	O	x
=	280340	262019	3594	7704	2560
I	17063	7877	213	2171	45
o	186	29	127	4	2
O	23950	4577	11097	522	4406
x	70646	3421	1717	152	33049

5 Experiments and evaluation

Data management of SkyServer is provided by MS SQL Server under Windows. To ensure comparability of the results we used the Windows installation for MonetDB5/SQL. The experiments ran on a machine with AMD Opteron 246 processor and 4GB RAM under Microsoft Windows Server 2003.

5.1 Data load

We ported the MyBestDR4 data set with approximate size of 1.5 GB as SQL Server database into MonetDB/SQL. The data was exported from the source DBMS through its bulk copy utility. Bulk loading of the tables carried the expected problems, such as encoding of specific values of string and text columns, and mismatch of some foreign key values. It took about 7 minutes where most of the time was spent in constraint checking. This is comparable with bulk import into MS SQL Server on the same machine which also took about 7 minutes.

The image size of the MyBestDR4 data set loaded in MonetDB is 850MB, which is approximately half the size of the MS SQL Server database. The reason is that redundant tables and covering indices are not stored at all, while the primary indices are stored only as a minimal sub-structure from which the entire indices are re-created on-the-fly when columns are touched for the first time.

5.2 Functionality

To evaluate the functionality provided by our implementation, we used two sources: the list of data mining queries in [6] (to be called a ’reference list’), and a subset of the query log. After developing some missing standard and MS-specific features, MonetDB/SQL can execute all the queries in the reference list, except query 12 that contains a direct call to a method from the HTM library which is not currently supported.

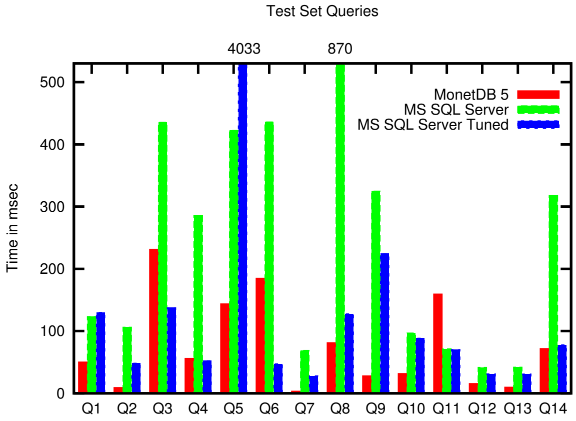

Figure 3. Elapsed times of the test set queries

We also ran a subset of 20K queries from the query log after we removed the queries irrelevant in our settings, such as queries retrieving server name or accessing SkyServer personal data storage MYDB. The subset was formed by randomly selecting a thousand queries for each of 20 days. SkyServer with MonetDB could execute 98% of this subset. The remaining queries utilize expressions in their GROUP BY clause, a feature supported in MS SQL Server, but not included in the standard.

5.3 Performance

In order to test the performance, we selected queries from the reference list(Q1-Q6), examples on the web site(Q7-Q9), and the most frequent patterns in the query log(Q10-Q14).

We made two measurements of the elapsed times in the reference platform: First in MyBestDR4 database which formed part of the distribution. Then a tuned database was used where a number of statistics and additional covering indices were created according to the recommendations given by the database engine tuning advisor run against the test set queries. The indices added about 100MB to the database size.

The results shown in Figure 3 are promising. The elapsed times of MonetDB/SQL implementation are comparable and often better than the reference system. The main reasons are as follows: the experimental data set fits entirely in main memory and thus no I/O is incurred by the queries. Furthermore, due to the column store of MonetDB, the execution engine touches only columns relevant to the query. For example, query Q7 from the test set shown bellow touches only 5 out of 59 columns of SpecObjAll table.

```
SELECT specObjID, z, zConf, SpecClass
FROM SpecObj
WHERE (SpecClass=fSpecClass('QSO')
   or SpecClass=fSpecClass('HIZ_QSO'))
   and z between 2.5 and 2.7
   and zConf > 0.90;
```

The execution plan in MonetDB checks the predicates on the individual columns and joins the intermediate results to form the final result set. The execution of the same query in MS SQL Server starts with a non clustered index scan that covers all columns but ZConf. To retrieve this last column, a clustered index seek is performed.

Longer elapsed times in MonetDB for some queries can be attributed to the following factors: due to the column store, join of columns is needed to produce the query result; MonetDB uses a sequential scan for range queries, in contrast to the wide use of B-trees in the reference platform; and finally, our optimizer is under development and sometimes produces sub-optimal plans. All these factors contribute to the slow execution of query Q11, illustrated in section 4.2. For example, the spatial access function scans the columns with spatial values, and the query touches 19 columns in the basic PhotoObjAll table which incurs corresponding joins to re-create the result records.

We noticed that queries for which MonetDB is faster than the initial MS SQL Server database, such as Q3, Q5, Q6, and Q8, performed a high number of I/O operations mainly due to non clustered index scan followed by clustered index seek. Those queries improved substantially in the tuned database, where appropriate covering indices existed for them. An interesting exception is query Q5, for which the index used in the tuned database covers nine out of ten columns and thus does not avoid clustered index seek needed to retrieve the last column. After manual re-creation of a fully covering index, the query became 8 times faster than in the initial database.

To summarize, the column store approach of MonetDB showed to be capable of providing performance comparable to the reference platform for the 1GB data set. Tables with hundreds of columns are not a big issue, taking into account that the majority of the queries touch a limited number of columns.

6 Outlook

The evaluation of the SkyServer application port and the observations from the query log raise the following questions: 1) how can we further improve the performance of the current MySkyServer on MonetDB platform; 2) how can we scale the application to the full 2TB size, and 3) can we provide high performance as required by data mining applications in astronomy research?

6.1 Improving performance

The measurements presented in the previous section were obtained using out-of-the-box MonetDB installation without any special tuning to the application. There are several directions to boost the performance further.

The observed relatively limited number of query patterns leads to the idea for a *workload-driven optimization* framework. Such a framework allows tuning the optimizer to the needs of a specific scientific application and saves on optimization time needed to support general query classes.

Analysis of the query log showed that many queries share the same pattern, differing only in constant values, or have substantial common sub-expressions. This observation hints at *multi-query optimization* as a promising approach to improve performance and increase query throughput.

The usage patterns observed in the manual interactions look like *telescoping the database* where zooming-in/ zooming-out is applied to the sky area of interest. In the script-based interactions, many queries retrieve the same sky location. This points our attention to materialization and re-use of the results of spatial access functions. Since MonetDB materializes all intermediate results, this translates into extending the query plan space to include re-use of materialized results and applying a caching advise policy that specifies which results to keep and for how long.

Important functionality in the application is implemented as SQL functions. In addition to materializing function results, in-lining of SQL functions needs to be investigated that would allow the SQL optimizer to perform a better job.

As it was shown for the queries with low performance, MonetDB currently uses sequential scan to support range-based queries. Although the scan of a single column in main memory has competitive performance for small to average size columns, there is a room for improvement by enhancing the MonetDB index structures. The emerging *cracking scheme*, which self-organizes based on data access frequency, seems particularly applicable[10, 7].

6.2 Partitioning and distribution

The current prototype works with 1% of the SkyServer data set that fits in the main memory of modern PCs. To handle the complete 2TB data set is a challenging task for our implementation, whose development was focused around efficient processing of data in main memory. To provide the scalability and speed-up required by the full-size SkyServer application, we currently investigate two directions: partitioning and distribution.

In a single server scenario we need a mechanism to localize and load into main memory the smallest possible data subset which is relevant to the query. In addition to the indexing techniques mentioned above, data partitioning can be used for maximum scale-up and speed-up. This means that, along with vertical fragmentation in the basis of MonetDB design, some form of horizontal fragmentation is needed. There are several alternatives for this. Horizontal data partitioning in record store systems is a well established research area whose results are used in commercial DBMSs. Given the limited number of query patterns, another possibility is to apply the workload-based categorical partitioning suggested for SkyServer database in [13]. The cracking scheme described above may also prove beneficial to generate partitions. We need to investigate how to efficiently combine these methods with vertical fragmentation, since processing of a partitioned column does not remove the need of joining the result with other, probably full-size, columns used in the query.

In a multi-server scenario setting, distribution and replication can provide further scale-up and speed-up. Distributed database technology has a long standing research tradition in business applications. This application domain had a strong influence on the techniques deployed for transaction management and query processing. The scientific domain, however, often has simpler transaction requirements. Timely propagation and non-destructive additions are more common than e.g. the real-time behavior of financial applications. Recent research on middleware on proactive measures in partially replicated databases could provide a technology bridge between both domains [12, 8].

6.3 Data mining

The SkyServer project bridged the first gap between the two fields of astronomy and database research. We made great progress in porting this large application to MonetDB/SQL, but aim at a much higher goal to help astronomers in their search for new astronomical events, objects, phenomena, etc. The unique features of MonetDB/SQL, such as main memory DBMS, vertical fragmentation and cache aware processing, will open a number of research directions currently not possible with the disk based system used by the SkyServer project.

One of the envisioned directions of research is astronomical data mining. It looks beyond single stars or galaxies, but tries to find correlations between astronomical events and to make classifications. One of the approaches is *cracking* the query tasks into a sequence of (interdependent) sub-queries to steer scientific data mining algorithms and to interactively explore science data. The sub-queries can be controlled by user interest, anticipated results, and the resource costs.

7 Conclusions

Scientific databases pose new challenges to the database community. Even the porting of an existing application to your favorite DBMS may take several months. Let alone if you start from a functionally incomplete SQL implementation.

In this paper we have reported on our experiences along this track. A fully functional implementation of MySkyServer on MonetDB/SQL has been realized. The performance figures show that a column store approach can provide competitive performance and is promising for the scientific domain.

Unlike the production system based on Microsoft SQL Server, the MonetDB open-source platform provides the necessary context for further research. It can be obtained from our project website http://monetdb.cwi.nl.

There are clear roads for in-depth studies and improvements in the way we support SkyServer. For example, the query zoom-in/zoom-out behavior opens up the road to develop algorithms for cache management utilizing this behavior for query optimization. The focus of attention gives proper handles to distribute and locally cache a small portion of the database. Usage analysis shows possibilities for improvements we want to investigate.

Acknowledgments

The authors would like to thank to the SkyServer team for delivering the data. This work was supported by Bsik Bricks program.

References

[1] S. Abiteboul, R. Agrawal, P. Bernstein, et al. The Lowell database research self-assessment. *Commun. ACM*, 48(5):111–118, 2005.

[2] P. Boncz, M. Zukowski, and N. Nes. MonetDB/X100: Hyper-Pipelining Query Execution. In *Proc. CIDR*, Asilomar, CA, USA, 2005.

[3] P. A. Boncz and M. L. Kersten. MIL primitives for querying a fragmented world. *The VLDB Journal*, 8(2):101–119, October 1999.

[4] J. Gray, M. A. Nieto-Santisteban, and A. S. Szalay. The Zones algorithm for finding points-near-a-point or cross-matching spatial datasets. *MSR-TR-2006-52*, April 2006.

[5] J. Gray, A. S. Szalay, A. R. Thakar, G. Fekete, W. O'Mullane, M. A. Nieto-Santisteban, G. Heber, and A. H. Rots. There goes the neighborhood: Relational algebra for spatial data search, April 2004.

[6] J. Gray, A. S. Szalay, A. R. Thakar, P. Z. Kunszt, C. Stoughton, D. Slutz, and J. vandenBerg. Data mining the SDSS SkyServer database. *MSR-TR-2002-01*, January 2002.

[7] S. Idreos, M. L. Kersten, and S. Manegold. Database Cracking. In *Proc. CIDR*, Asilomar, CA, USA, January 2007.

[8] L. Irún-Briz, H. Decker, R. de Juan-Marín, F. Castro-Company, J. E. Armendáriz-Iñigo, and F. D. Muñoz-Escoí. Madis: A slim middleware for database replication. In *Euro-Par*, pages 349–359, 2005.

[9] M. L. Kersten. Database architecture fertilizers: Just-in-time, just-enough, and autonomous growth. In Y. E. Ioannidis, M. H. Scholl, et al., editors, *EDBT*, volume 3896 of *Lecture Notes in Computer Science*, page 1. Springer, 2006.

[10] M. L. Kersten and S. Manegold. Cracking the Database Store. In *Proc. CIDR*, pages 213–224, Asilomar, CA, USA, January 2005.

[11] T. Malik, R. Burns, and N. Chawla. A black-box approach to query cardinality estimation. In *Proc. CIDR*, Asilomar, CA, USA, January 2007.

[12] E. Pacitti, C. Coulon, P. Valduriez, and M. T. Özsu. Preventive replication in a database cluster. *Distributed and Parallel Databases*, 18(3):223–251, 2005.

[13] S. Papadomanolakis and A. Ailamaki. Autopart: Automating schema design for large scientific databases using data partitioning. In *SSDBM*, pages 383–392. IEEE Computer Society, 2004.

[14] Sloan Digital Sky Survey, http://www.sdss.org/.

[15] Sloan Digital Sky Survey / SkyServer, http://cas.sdss.org/.

[16] M. Stonebraker, D. J. Abadi, A. Batkin, et al. C-Store: A Column-oriented DBMS. In *Proc. VLDB*, Trondheim, Norway, 2005.

[17] A. R. van Ballegooij, R. Cornacchia, A. P. de Vries, and M. L. Kersten. Distribution rules for array database queries. In *Proceedings of the International Workshop on Database and Expert Systems Application*, pages 55–64, Copenhagen, Denmark, August 2005.

[18] M. Zukowski, S. Héman, N. Nes, and P. Boncz. Superscalar RAM-CPU cache compression. In *Proceedings of the International Conference of Data Engineering (IEEE ICDE)*, Atlanta, GA, USA, 2006.