MonetDB, a novel spatial column-store DBMS

Maarten Vermeij[1], Wilko Quak[1], Martin Kersten[2], Niels Nes[2]

[1] TUDelft, OTB, section GIS-technology, The Netherlands

c.w.quak@tudelft.nl,m.j.vermeij@tudelft.nl

[2] CWI Amsterdam, The Netherlands

Niels.Nes@cwi.nl, Martin.Kersten@cwi.nl

Abstract

Column-store database engines are a promising track in database research to handle data warehouses. In this paper we describe our experiences in extending the open-source database management system MonetDB with geo-spatial functionality. The approach taken is to leverage the existing geo-spatial software library GEOS through the extensibility features of this DBMS. The result is a high-performance solution using a software stack that enables future research and development improvements in many directions. In our paper we first give an overview of the MonetDB architecture then we describe how this architecture is beneficial for the handling of spatial data.

1 Introduction

Recent years have seen a flurry of activities in the database research arena, aimed at improved processing against large data warehouses. The trend is to focus on analysis applications, which call for a different organization of the database storage layers. In particular, the orientation on columns as the prime storage element has become fashionable.

One high-performance open-source column-store is MonetDB. It has been successfully deployed in application areas ranging from data mining, OLAP, information retrieval and multimedia data management. In many warehouse applications, MonetDB achieves a 10-fold raw speed improvement for SQL and XQuery over the competitor RDBMSs (see: `monetdb.cwi.nl`). MonetDB achieves its goal by innovations at all layers of the DBMS, e.g. the column-store approach, a

storage model based on vertical fragmentation, a modern CPU-tuned query execution architecture, automatic and self-tuning indexes, run-time query optimization, and a modular software architecture.

We embarked on a project to unleash MonetDB's performance also in the area of spatial applications. For this we used the existing open source GEOS libraries (`http://trac.osgeo.org/geos/`), also used by PostGIS, and make it available via the MonetDB/SQL engine. Using an already existing library speeds up the integration process significantly. There was no need to re-invent and re-implement all spatial functions.

In this paper we describe how MonetDB performs as a spatial engine and how the design characteristics of MonetDB helped to solve problems that are hard to crack in other spatial DBMS systems. The main characteristics of our spatial-enhanced version of MonetDB are:

- The vertical fragmentation (column-store approach) used by MonetDB to store its data is very beneficial for spatial query processing. The prime reason is that with spatial filter techniques only a fraction of the geometries of a table are really needed in most cases. In a traditional tuple based storage model, the geometry data is still in the way. Due to the size of the geometry only a few tuples can be stored in a disk-block, this means that almost each tuple accessed in a query results in a disk-block access. Vertical fragmentation ensures that the non-needed attributes are not in the way.

- Spatial queries are often very hard to deal with. Often the query optimizer needs to be helped by the pragmas or hints to the SQL optimizer. We will indicate how to leverage the self-organizing capabilities of MonetDB software infrastructure for spatial data.

- In the spatial domain XML (and specifically GML) is becoming more and more used. However, querying a large XML document is still cumbersome. MonetDB has a proved performance track record as a XQuery engine. Combining the efficient XQuery functionality of MonetDB with the spatial module leads to and an powerful GML processing solution.

In the spatial DBMS field the last years focus has been on integrating support for spatial data inside existing commercial and open-source products with Oracle and PostgreSQL/PostGIS as forerunners and recently Microsoft added spatial support to SQLServer. A nice research article on spatial DBMSs dates from 1997 (Patel et al. 1997).

The remainder of the paper is organised as follows. In Section 2 we sketch the MonetDB architecture. In Section 3 we describe how the characteristics of MonetDB help in making an efficient spatial DBMS engine. Finally in Section 4 we give a short overview of our future plans with MonetDB spatial.

2 MonetDB Architecture overview

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 (Boncz & Kersten 1999, Stonebraker et al. 2005). In this field, relational tables are broken vertically with each column representing a single relational attribute. Almost as if each column is stored in a separate table or even ordinary array, but then with an implementation that is geared to optimally exploit this structure. This approach leads to a much simplified system architecture and opens many routes to increase performance. 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 for statistical analysis. 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. Currently available front-ends provide an SQL and an XQuery interface to the database server. The server is addressed in a proprietary language, called MonetDB Assembly Language(MAL). MAL is a relational algebra language that 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 the 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 front-end parses SQL queries and compiles them into semi-optimized MAL plans which exploit 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 documentation at `http://monetdb.cwi.nl/`.

3 MonetDB design principles and impact on spatial data

In this Section we describe how the design principles of MonetDB are of great benefit for the handling of spatial data. The relevant principles are introduced in the sections below:

3.1 Column-oriented data storage

Vertical fragmentation or column-oriented data storage is beneficial for spatial query processing. In a traditional tuple based storage model, where all data for a tuple is stored physically together, the geometry data is still in the way. The size of the geometry implies that only a few tuples can be stored in each disk-block. This means that almost every tuple in a query answer results in a disk-block access, even if the geometry is not needed for answering the query. Vertical fragmentation ensures that the non-needed geometries are not in the way.

Filtering can be further improved readily by storing multiple approximations in the same column. In terms of a traditional relation model, each spatial attribute comes with several approximations to ease filtering.

Some geometry types, for example polygons can become very complex. Complex geometries require both a larger storage size as well as a more computing time in many analyses. To speed up e.g. filtering of polygon geometries within a table, the filter-refine schema as described in (Kriegel et al. 1993) can be used. This system uses approximate geometries such as the minimum bounding rectangle, minimum bounding circle and convex hull to allow a fast reduction in the number of candidate geometries in a spatial query(conservative approximation). Besides this minimum outside bounding approximations to reject geometries, it is also possible to use maximum enclosed geometries, e.g. maximum enclosed circle and maximum enclosed rectangle, to quickly identify definite positives without calculations on the actual geometries (progressive approximations). These approximate geometries could be stored in hidden tables that are automatically used by MonetDB when performing spatial queries. Since columns are stored independently, having more columns does not adversely affect performance on queries that use only the original columns. This set-up behaves more or less like an index in a standard (row-based) relational database.

Another benefit of the column-store approach is that the storage of each column can be optimized towards the access characteristics of the specific column type without side-effects on the performance of other columns. For example, specific access structures are easier to realize, and (run-length, front-, rear-) compression schemes can be more readily applied.

The implementation of the spatial algorithms does not have to worry about circumstantial data items and access profiles. This also opens the door to future enhancements of the support for spatial types in MonetDB, such as specific compression schemas for spatial columns (Isenburg et al. 2005).

The column-store also comes at a cost. Many joins are needed to reproduce the original table. Although this join operation is highly optimized in MonetDB, it is the moment where you pay for the vertical fragmentation. Despite the cost, there are several arguments for the introduction of vertical decomposition:

- Databases tend to grow wider and get tables with more and more columns. In most cases only a few of these columns are used in the where-clause of a query and the rest is just dead weight in the selection of records. Column-store DBMSs a very good at querying tables with lots of columns since only the columns relevant for the query need to be accessed.

- DBMSs continue grow in size, but the amount of information a human can process does not change much. Since the display of all columns for browsing is one of the few reasons to retrieve all columns (the `select *` operation) the relative cost of the final join will be smaller.

3.2 Optimizing queries with spatial data

Over the years several attempts have been made to create a retargetable or modular query optimizer. The more promising ones are based on term rewriting, which provides a setting to reason on its correctness(Becker & Güting 1992). It is, however, also known that many rewrites depend both on the inherent semantics of the query language and circumstantial information, such as availability of indices, algorithms and transaction protection level. In these cases, the rule rewriter quickly becomes difficult to track and keep consistent.

A query optimizer is often a large and complex piece of code, which enumerates alternative evaluation plans from which 'the best' plan is selected for evaluation. Limited progress has been made so far to decompose the optimizer into (orthogonal) components, because it is a common believe in research that a holistic view on the problem is a prerequisite to finding the best plan. PostgreSQL supports steering the optimizer using global variables. Conversely, commercial optimizers often use a cost-model driven approach, which explores part of the space using a limited number rewriting rules. More recently, database engines rely on workload analysis using query logs to learn.

The MonetDB software stack opens up this box of Pandora, by providing an easy scheme to debug, deploy, and trace optimizers geared at well-defined tasks. Our hypothesis is that query optimization should be realized with a collection of code transformers, each targeted at a specific task, and dynamically activated.

inline	remap	evaluate
costModel	coercions	empty set
access modes	aliases	merge tables
common terms	accumulators	
deadcode	reduce	garbage collector
dataflow	multiplex	

Figure 1: The MonetDB/SQL optimizer pipeline

The MonetDB distribution comes with a large collection of optimizer modules[1] They are developed up to the point that they could be used in production code or to experiment with the optimizer software infrastructure. They are highly targeted to a particular problem. Figure 1 shows the modules forming the optimizer pipeline for SQL queries.

The effectiveness of the optimizer toolkit is illustrated by its code size. Each well-defined optimizer task just takes a few pages of C-code, relying on a small library of generic support routines to analyse and manipulate the MAL internal representation. To summarize the generic functionality:

Functional behaviour A critical property for optimization is to easily recognize operators with and without side-effects. A large collection of such harmful operators can be recognized by their VOID type, for any operator that does not produce a result need not be executed. A VOID returning operator thus should have effect elsewhere. Other operators are easily recognized by the module name or an explicit property specified with the function definition.

Lifespan analysis All variables have a lifespan, denoted by properties *beginLifespan*, i.e. the statement where it receives its first value, and *endLifespan*, i.e. the statement where it is last used. If its last use lies within a BARRIER block, then its *endLifespan* is aligned with the block EXIT.

Flow analysis In many optimization rules, the data flow dependency between statements is of crucial importance. The MAL language provides a multi-source, multi-sink dataflow network. Optimizers typically extract part of the workflow and use the language properties to enumerate semantic equivalent solutions, which under a given cost model turns out to result in better performance.

Static evaluation Some statements are independent of the execution context. In particular, expressions over functions without side-effect and constant arguments can be evaluated before the program block is considered further.

[1] See http://monetdb.cwi.nl/projects/monetdb/MonetDB/Documentation/The-MAL-Optimizer.html for a complete overview.

Pattern replacement A major task for an optimizer is to select statement (sequences) that can and should be replaced with cheaper ones. The cost model underlying this decision depends on the processing stage and the overall objective.

This setup provides an excellent stepping stone for developing spatial oriented optimizers. They can be tested in isolation and their interference with other optimizers is easily determined using the debugging tools provided. We refer to the documentation and code base for more details.

3.3 Linkage between XML and spatial

One of the strong points of MonetDB is its capability to store XML (Boncz et al. 2006). Currently the linkage between SQL/XML integration is under development, therefore the example in this paragraph does not work in the current version. However, all components described have been proven in practice.

Expressing a spatial where clause as an XQuery expression on a GML geometry is not very efficient. However, the integration of the spatial datatypes in MonetDB with the XML types makes elegant and efficient solutions possible. The following sample script demonstrates the loading of a geometry attribute from a GML file.

```
COPY INTO buildings(XMLgeometry) FROM '/tmp/buildings.gml'
   DELIMITER 'buildings(geometry)';
ALTER TABLE buildings ADD COLUMN geometry polygon;
UPDATE buildings SET geometry = PolygonFromGML(XMLgeometry);
```

If the GML file contains a FeatureCollection of building objects, the first instruction copies the GML document into MonetDB and breaks it into pieces. This shredding can be done in many different ways depending on the application. Here we perform a top-down parse of the GML document where the XML-subtree geometry is extracted. Now we add a spatial column to the building table and convert the XML geometry object into a geometry. We can use this geometry attribute for efficiently accessing the spatial component of the data in combination with XPath for the rest of the document.

4 Concluding remarks and future plans

In this paper we have shown that we have successfully extended MonetDB with spatial functionality. The result is an exciting product that has a lot of potential for further research. Some of the ideas we have for the future are:

- Finalize the integration between the spatial model and the XML storage.
- Implement a host of approximations on polygons to fully exploit the filter-refine process.
- Improve query optimization on mixed spatial/non-spatial queries.
- Write specific compression schemes for spatial columns to enable compact storage of spatial columns.
- Create plug-ins for several platforms that handle spatial data (MapServer, ...)

References

Becker, L. & Güting, R. H. (1992), 'Rule-based optimization and query processing in an extensible geometric database system.', *ACM Trans. Database Syst.* **17**(2), 247–303.

Boncz, P. A. & Kersten, M. L. (1999), 'MIL primitives for querying a fragmented world', *VLDB Journal: Very Large Data Bases* **8**(2), 101–119.
URL: *citeseer.ist.psu.edu/boncz99mil.html*

Boncz, P., Grust, T., van Keulen, M., Manegold, S., Rittinger, J. & Teubner, J. (2006), MonetDB/XQuery: a fast XQuery processor powered by a relational engine, *in* 'SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data', ACM, New York, NY, USA, pp. 479–490.

Isenburg, M., Lindstrom, P. & Snoeyink, J. (2005), 'Lossless compression of predicted floating-point geometry', *JCAD - Journal for Computer-Aided Design* **37**, 2005.

Kriegel, H.-P., Brinkhoff, T. & Schneider, R. (1993), 'Efficient spatial query processing in geographic database systems', *Data Engineering Bulletin* **16**(3), 10–15.
URL: *citeseer.ist.psu.edu/kriegel93efficient.html*

Patel, J., Yu, J., Kabra, N., Tufte, K., Nag, B., Burger, J., Hall, N., Ramasamy, K., Lueder, R., Ellmann, C., Kupsch, J., Guo, S., Larson, J., Witt, D. D. & Naughton, J. (1997), 'Building a scaleable geo-spatial DBMS: technology, implementation, and evaluation', *SIGMOD Rec.* **26**(2), 336–347.

Stonebraker, M., Abadi, D. J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., ONeil, E., ONeil, P., Rasin, A., Tran, N. & Zdoni, S. (2005), C-store: A column-oriented DBMS, *in* 'Proceedings of the 31st VLDB Conference', pp. 553–564.
URL: *http://www.vldb2005.org/program/paper/thu/p553-stonebraker.pdf*