Building a Scalable Spatial OLAP System*

Oliver Baltzer
FlagstoneRe
Suite 700, 2000 Barrington St
Halifax, Nova Scotia, Canada
obaltzer@flagstonere.com

Andrew Rau-Chaplin
Faculty of Computer Science
Dalhousie University
Halifax, Nova Scotia, Canada
arc@cs.dal.ca

Norbert Zeh
Faculty of Computer Science
Dalhousie University
Halifax, Nova Scotia, Canada
nzeh@cs.dal.ca

ABSTRACT

We present a modular design of a spatial OLAP system. Contrary to most previous work on spatial OLAP systems, which combine existing OLAP systems and geographic information systems using a middleware layer, our system directly integrates OLAP and the processing of spatial data and, thus, provides the full power of spatial OLAP without performance penalty and allows a unified treatment of spatial and non-spatial data.

Keywords

Spatial OLAP, Pipelined query evaluation, parallelism

1. INTRODUCTION

With an increasing number of applications that store spatial and non-spatial data in data warehouses, the development of efficient and scalable spatial OLAP systems, which combine traditional OLAP capabilities with support for spatial queries and for OLAP operations over spatial dimension hierarchies, is increasing in importance.

Most previous work on the design and implementation of spatial OLAP systems focused on the integration of existing OLAP and GIS components into a single application using specialized middleware software [2, 3]. While this middleware approach is often cost effective and quick to implement, the challenge with spatial OLAP queries using such systems is that the OLAP component is not designed to handle spatial attributes while the GIS component is not designed to support OLAP operations at all. However, the implementation of complex OLAP operations over spatial dimensions requires a constant flow of data between the two components, and thus the middleware layer becomes a major bottleneck.

In this paper, we present a design for a spatial OLAP system that integrates the processing of spatial dimensions directly into the OLAP system. The result is a system that is efficient and scalable and at the same time provides the full expressive power of OLAP over both spatial and categorical data dimensions. Our system decomposes each spatial OLAP query into well delineated tasks (retrieval, join, aggregation, etc.) and provides components, which we call *mini-engines*, to carry out these tasks. The mini-engines involved in evaluating the query exchange their data through data streams and process data in parallel by leveraging multiprocessor and multicore systems.

*Research Supported the Natural Science and Engineering Research Council of Canada.

SAC'13 March 18-22, 2013, Coimbra, Portugal.

2. PIPELINED EVALUATION OF SPATIAL OLAP QUERIES

The pipeline approach for the evaluation of database queries was initially proposed by Boral and DeWitt in 1980 [1] and has been adopted in a number of database engines and in traditional OLAP systems that have grown out of such database management systems. The fundamental idea of the pipeline model is to divide the evaluation of a complex query into a number of simpler tasks, each an independent component consuming a set of input data and producing some set of output data.

Our pipeline query evaluation model is derived from this basic pipeline approach and can be applied to spatial and non-spatial OLAP queries involving both spatial and categorical dimensions using a single integrated framework. In our model, we refer to the components responsible for the execution of tasks as *mini-engines*, to reflect that their role is to act autonomously as part of a larger query evaluation engine. Data is transferred between mini-engines using data streams, and each mini-engine can pass its results on to the next as soon as they become available. This stream-oriented approach to data transfers allows an interleaving of the execution of the individual mini-engines and the concurrent execution on multiple processors or processor cores. It is also often the case that a given query can be answered using different combinations of mini-engines, which allows a query optimizer to choose the combination that is most efficient depending on characteristics of the data set.

For mini-engines to be interoperable, the interfaces between them have to be well defined. To define these interfaces, mini-engines that have a similar purpose (e.g., indexed storage) but varying implementations (e.g., B-tree or hash table) are grouped into a class for which the interfaces to other classes of mini-engines are defined. These interface definitions provide the rules by which mini-engines can be combined to obtain evaluation engines for complex queries.

The abstraction of query evaluation tasks into mini-engines allows for the encapsulation of their implementation. This

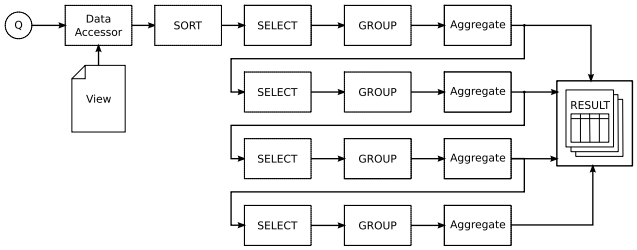

Figure 1: Query decomposed into mini-engines.

allows the development of specialized mini-engines that are particularly efficient for certain types of queries and can replace more general-purpose ones as long as they use the same interface. More importantly, it allows new mini-engines to be added to the system as necessary to efficiently answer new types of queries. Thus, the system is easily extensible.

Our pipeline model defines 8 classes of elementary mini-engines which are sufficient to evaluate a wide range of spatial OLAP queries: DATA ACCESSOR, SELECT, FILTER, GROUP, AGGREGATE, JOIN, SORT, and RESULT STACK.

To illustrate the mini-engine concept, consider the following abstract standard OLAP query:

```
SELECT     a1, a2, a3, AGGREGATE( f )
FROM       View
GROUP BY ROLLUP( a1, a2, a3 )
```

It retrieves all records from view View and performs aggregation on the facts attribute f for each group of records w.r.t. the attribute sets provided in the GROUP BY clause. In this example the ROLLUP function generates the attribute sets {a1, a2, a3}, {a1, a2}, {a1}, and {}, each representing a separate group-by aggregation. The results from each group-by aggregation are then merged into a combined result view. Note, this query does not specify the data types of attribute dimensions a1, a2, and a3 or fact dimension f. In fact, in the context of a spatial OLAP system the attribute dimensions my be a combination of categorical and spatial dimensions and the fact dimension may be a spatial measure.

An evaluation strategy for this query using the pipeline model is shown in Figure 1. The data flow graph in this figure shows the decomposition of the query evaluation into smaller tasks carried out by different mini-engines, and the flow of data between these mini-engines.

The first step in the evaluation of our example query is the retrieval of all records from the input view using the DATA ACCESSOR mini-engine. Generally, the DATA ACCESSOR mini-engine provides an abstraction of one or more non-linear data structures that provide efficient access to the records in the view matching a given query Q. In this example, no such query Q is specific and all records are to be retrieved from the view.

The records retrieved by the DATA ACCESSOR are passed to the next mini-engine in the data flow graph, a SORT mini-engine in this case. This mini-engine rearranges the input records in a particular order. In this example, the records are sorted according to the grouping set specified in the GROUP BY clause. Since sorting is an expensive operation it is important to exploit the ordering of the input records, if any, to simplify this sorting step. To facilitate this, the metadata associated with each stream includes the sort order of the records in the stream, i.e., the set of attributes by which the records are guaranteed to be sorted.

In the next step, the SELECT mini-engine selects from the incoming record stream only those record attributes that are relevant for the further processing of the query. In the first instance of the SELECT mini-engine for this example query, the attributes a1, a2, a3, and f are selected, and a new record stream containing only these attributes is emitted. A GROUP mini-engine is then used to obtain a grouping of the records in the data stream, which corresponds to the first most detailed level of aggregation groups. The grouped records are then passed to an AGGREGATE mini-engine, which performs the actual aggregation for each group.

Note that the ROLLUP statement generates a number of grouping sets, each representing a different level of aggregation, which makes it necessary to repeat this sequence of SELECT, GROUP, and AGGREGATE mini-engines for each aggregation level. Since the ROLLUP function constructs grouping sets in such a way that each grouping set is a subset of the previous grouping set, while the attributes remain in the same order, it is possible to use the aggregated results from the previous grouping set as an input for the next grouping set. This property is used to speed up the query, and the output stream of each aggregation step is used both as part of the output collected by a RESULT STACK mini-engine and as input to the next level of aggregation.

The reference implementation of our pipeline model for evaluating spatial OLAP queries demonstrates our model can be realized in a fully functional system that exploits multicore parallelism to be fast and scalable. The implementation uses primarily Python as implementation language, but also relies on C/C++ for some external libraries. We expect a carefully tuned implementation in pure C/C++ to exhibit significantly better performance. Even so, already our Python prototype significantly outperformed PostgreSQL's spatial extension in our experiments.

Our experiments show that the reference implementation can successfully utilize multiple processor cores during query evaluation. The speed-up in query evaluation time grows roughly linearly with the number of processors and with efficiency above 85% for up to 8 processor cores.

In a direct comparison between our reference implementation and PostgreSQL with its spatial extension PostGIS, we observe that our implementation outperforms a carefully hand-tuned query implementation using PostgreSQL's native PL/pgSQL language by a factor 3.1. In comparison to an automatically optimized implementation using standard SQL constructs to calculate only the lowest roll-up level, our reference implementation is able to outperform PostgreSQL by a factor of more than 6.

3. REFERENCES

[1] H. Boral and D. J. DeWitt. Design considerations for data-flow database machines. In *SIGMOD*, 1980.

[2] E. Malinowski and E. Zimanyi. Implementing spatial data warehouse hierarchies in object-relational DBMSs. In *Int. Conf. on Enterprise Info. Sys.*, volume 7, 2007.

[3] S. Rivest, Y. Bédard, M.-J. Proulx, M. Nadeau, F. Hubert, and J. Pastor. SOLAP technology: Merging business intelligence with geospatial technology for interactive spatio-temporal exploration and analysis of data. *ISPRS Journal of Photogrammetry & Remote Sensing*, 60:17–33, 2005.