RAM: array database management through relational mapping

van Ballegooij, A.R.

Publication date
2009

Chapter 1

Introduction

Database technology is a central component in today's information technology. All kinds of business applications are built around central data repositories managed by advanced database management systems, and for good reasons. The primary selling point of database management systems (DBMS) is the potential for reduced application-development time by using the data-management features made available. But there are other important benefits from a business point of view: A central database allows organizations to enforce a standard way of organizing and managing data and it aids in keeping data available to various applications up-to-date and consistent.

At present the (commercial) database world is dominated by relational database management systems. Relational database technology replaced flat-file storage systems because its high level of abstraction separates application code from physical storage schemes. The relational data model, introduced by Codd in 1970 [1], models data by grouping related objects into distinct relations.

Oddly enough, database technology has not penetrated scientific computing in the same way it has the business world. In the world of supercomputers and large-scale networks of computers, grids, custom-built software solutions are omnipresent. Yet, scientific instruments and computer simulations create vast volumes of data to be organized, managed, and analyzed: these are the primary tasks of a database management system. The lack of acceptance of (relational) database technology in science can be attributed to a number of issues: the lack of performance offered by existing database management systems; the mismatch between scientific paradigms and the relational data model; and the unclear benefit of the investments required to switch from existing application frameworks, that at present suffice, to a database-driven environment. The common preference to develop applications in 3rd-generation languages (C++, FORTRAN, Matlab, ...) directly can only be changed by convincingly showing that the other problems can be solved.

1.1 Large Data Volumes

Scientific data sets are growing into the petabyte (10^{15} bytes) range and clearly pose a data-management challenge. Database technology capable of storing and managing these volumes exists and is in use, for example at CERN [2]. But the ability of a database management system to store vast volumes of data does not guarantee that it can perform complex analysis tasks efficiently as well. The (perceived) lack of efficient data-processing capabilities in database systems has resulted in many databases systems merely functioning as a persistent store while external application programs perform analysis tasks.

The process of configuring a database management system for maximum performance is known as database tuning. This problem is known to be difficult as it involves carefully balancing many parameters [3]. And since scientific workloads are non-typical for relational systems, they require non-standard DBMS configuration. Costly features, essential in the business domain, may not necessarily be required in a scientific setting. For example, traditionally database management systems have offered fine-grained transaction and recovery control to retain as much data as possible in case of system failure: For many scientific analysis tasks simpler, and thus computationally cheaper, solutions may suffice, like recovery through re-computation of certain analysis results.

Properly tuned current state-of-the-art database technology showcases respectable processing power. This processing power is for example shown by the publicly available results for the TPC-H benchmark, a simulation of decision-support systems that examine large data volumes with complex queries. In this scenario, current database technology can efficiently process complex queries over datasets into the multiple terabyte (10^{12} bytes) range [4]. Although the TPC-H benchmark results do not yet showcase petabyte-scale processing capabilities, current trends in business, such as the need to analyze rapidly growing telecommunications and logistics logs, are driving database technology developments to handle ever larger data sets.

Meanwhile, typical relational query processing techniques are independently making their way into high performance computing systems. For example, the Google map-reduce technique applies the inherent parallelism in set-oriented bulk processing of data to parallelize complex analysis tasks over thousands of computers [5]. This technique is similar to those used to push the performance envelope of distributed database technology [6]. At a lower level, basic linear-algebra operations at the core of many scientific computing problems have been shown to benefit from data abstraction. For example, by utilizing generic relational data access methods such as join algorithms, matrix operations over complex storage schemes can be accelerated [7].

Trends in the evolution of database technology are addressing the challenges posed by very large scientific data sets [8].

1.2 Multi-Dimensional Arrays

What remains is the interface hurdle imposed by the mismatch between computational paradigms and the relational model, generally known as the impedance mismatch. While the relational data model is adequate for storing and analyzing scientific objects, implementing the algorithms required on top of a relational interface is often awkward and cumbersome. Database management systems do offer rudimentary support for certain types of scientific data, such as spatial data and time series, but have not supported the multi-dimensional array as a core data type. The absence of multi-dimensional arrays as a primary data type in relatinal database systems is the main driving force behind the development of specialized storage libraries for scientific applications such as NetCDF [9], and has been argued to be the essential ingredient required for database technology to be embraced by the scientific community [10].

The current standard for database query languages, SQL-99 [11], does not offer primitives to construct or query bulk data arrays. Arrays in SQL-99 are small-scale structures that allow collections inside a single attribute, such as several lines of text that form an address. Bulk storage in SQL remains fully relational, but there have been several attempts at the development of database technology centered around the multi-dimensional array structure.

Multi-dimensional array support for databases is often studied from a theoretical perspective, with a focus on query-language design and high-level optimization strategies rather than data-management issues. For example, the array query language [12] (AQL) has been an important contribution toward the development of array support in database systems by proposing a generic array-comprehension query-language that seamlessly integrates into an existing set-based data model. Unfortunately, the main contribution remains theoretical, as it has not evolved beyond a prototype system. Alternatively, the array manipulation language [13, 14] (AML) shows potential for interesting optimizations of array queries made possible by restricting the query-language. Likewise, this system has not evolved beyond a prototype capable of handling several specific cases.

One example of a complete system is the RasDaMan system [15]. It is primarily an image database management system, but showcases many of the features required for a generic array database system. It consists of an efficient storage manager [16] and an array oriented query language – RasQL – implemented in a frontend that can interface with object-oriented and relational database systems. Work by Sarawagi et al. [17] takes this approach one step further by adding support for large multi-dimensional arrays to the relational POSTGRES database system [18]. Here, multi-dimensional arrays are stored in specialized data structures that are integrated into the core of the database system. The focus of this work is on the low-level storage issues of large arrays.

The commonality among these array-database efforts is that all of them have been realized through custom, array-specific, functions, either implemented through extension of existing database systems or as stand-alone prototypes.

1.3 Relational Mapping

An alternative to implementation of new database functionality through new native functions is relational mapping: translation of operations over non-relational data to relational queries over a relational representation of that data. This approach has been used in object-relational database solutions where object-oriented database functionality is realized by mapping operations to a relational DBMS [19]. Most functionality required to support an object-oriented interface on top of a relational database system is readily available in the relational paradigm. Features that require additional functionality, such as user defined types and functions, have been implemented in mainstream database systems [20] and have been standardized and included in the SQL standard [11].

The object-relational approach effectively creates a new interface to an existing database management system, which allows object-oriented data and relational data to be combined in a single framework. As this approach delegates data-management to the relational database system, data-management functionality readily available can be reused for object-oriented data. Additionally, the different access paths to the data, object-oriented and relational, combine the best of both worlds: The object-oriented interface simplifies applications by encapsulating database interaction in persistent objects, while the collected data can be bulk-processed for analysis using the relational access path.

Despite the advantages of an object-relational mapping approach, specialized implementations of object-database systems exist. The argument for a native implementation is performance: A native implementation avoids the inevitable overhead introduced by the mapping process. It is common belief that object oriented database systems are well suited for applications involving complex and heavily interrelated data. The relational representation of such complex entities is "flattened", e.g. see [21], and putting these entities back together in a meaningful way requires joining and sorting, both costly operations [22]. Queries with multiple multi-way joins, required to reconstruct complex objects, are a problem for relational database management systems [23].

Following the success of the object-relational approach, the emergence of XML databases and the XQuery language [24, 25] has lead to various XML-relational mapping schemes, for example [26, 27, 28]. Relational storage of XML data is based on "shredding"; this process translates a tree-based XML document into (several) relational tables.

The semi-structured XML tree is inherently associated with a navigational processing paradigm. Native XQuery implementations, implementations based on a tree representation of the data, tend to follow this navigational paradigm explicitly. Conversely, the relational paradigm supports bulk processing of data, which can be leveraged by XML-relational approaches, for example by detecting opportunities to use optimized join algorithms [29]. The differeces between these paradigms have resulted in a situation where native implementations outperform XML-relational systems for

simple queries over small data sets while relational approaches tend to be significantly more efficient at handling complex queries over large data sets [30].

Efficiency issues aside, one of the main advantages of a relational mapping approach is that support for new data types is automatically integrated into the existing relational framework. This integration is not standardized as the details of the mapping scheme differ per implementation, but since data is stored in relations it is relatively easy to combine foreign and relational data in queries.

1.4 Research Objectives

We propose an approach similar to the object-relational and XML-relational schemes for multi-dimensional array data: the Relational Array Mapping (RAM) system. **The research objective of this thesis is the realization of an extensible array database architecture using relational mapping and existing relational database technology.** Throughout this thesis, the research objective is addressed through the three separate goals outlined in the remainder of this section by discussing the design of the RAM system as visualized in Figure 1.1.

The RAM system is isolated in a front end that implements the relational mapping of the multi-dimensional array data and query language. This way, the RAM front-end operates alongside existing front-ends, such as a SQL or XQuery compiler, that access the same database system. The concept of multiple access methods to the same database system is a classical database management system design pattern introduced in the System-R architecture [31]. The bulk of the research regarding the RAM system is conducted in the context of the MonetDB database system [32], which is designed to be easily extensible through this multiple-front-end pattern. It is possible that certain functionality required for (efficient) array query processing is not readily available in a given relational back-end, therefore the design allows room for the addition of specialized array functionality in the relational back-end itself.

The first goal is the specification an efficient array mapping scheme. Chapter 3 presents an array-oriented data model and shows how this data model can be implemented in a relational environment.

The front-end consists of four separate components that each perform a distinct step in query translation: the first component normalizes the queries for further processing, the second component translates the normalized queries to an intermediate array algebra, the third component optimizes array queries through rewriting of the algebraic array-expressions, and finally the fourth component translates the query to the language of the relational back-end. Explicit separation of these components allows a study of each of these query translation processes in isolation. Additionally, it isolates functionality related to a specific back-end in a single component, which facilitates the support of a variety of back-ends by replacing this single component.

The second goal is to explore the benefit of optimization at the array expression level in addition to relational query optimization of translated array queries.

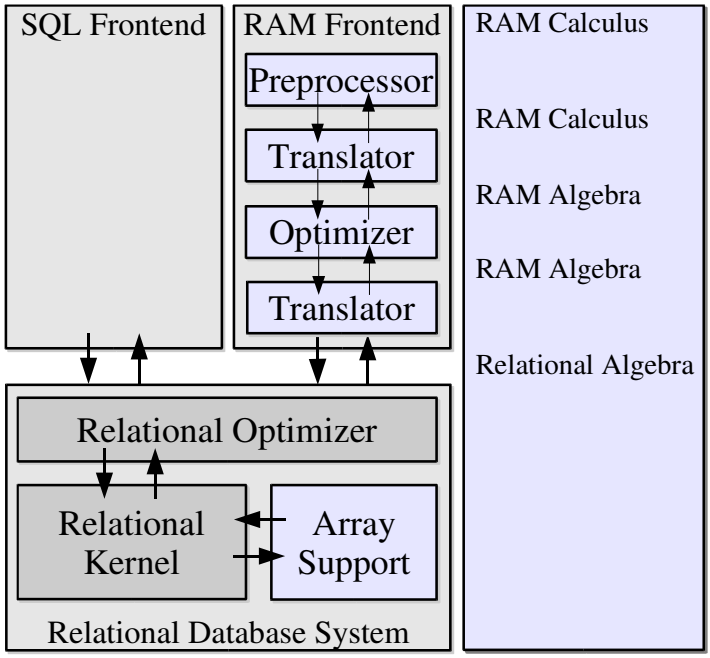

Figure 1.1: System architecture

Chapter 5 explores the suitability of traditional relational optimization techniques to be applied to the intermediate array algebra. Optimizing the array algebra expressions is similar to the logical optimization phase in relational optimizers – the best order of operations is chosen without fixing which physical operator implementations are to be used – and the focus is on applying known techniques from the relational domain to arrays.

The third goal is to show that translation of array operations directly into primitive relational operations allows for more efficient execution than high-level relational query languages would. We explore the specifics of RAM translation to several back-ends in Chapter 4 and discuss the merits of generating a 'smart' physical relational query plan directly from RAM rather than relying on the back-end to optimize naively generated query plans. Similar to the physical optimization phase in relational optimizers: The best physical operations are chosen to evaluate an optimized logical plan given known properties of the data to be processed.

The applicability of the system in applications and the benefit of the query optimization are evaluated in Chapter 6. There we present a case study using the RAM

system and several experiments that showcase the effectiveness of different optimization techniques. For these experiments we focus on multimedia (retrieval) as an application domain.

The remaining chapters of this thesis, Chapter 2, and Chapter 7, respectively anchor this work in literature and wrap up the thesis. Chapter 2 presents an investigation into existing database technology and its relation with scientific computation and array processing in particular. And this thesis is concluded in Chapter 7 by a summary of the results presented and a discussion of its contributions.

Bibliography

[1] E.F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[2] Dirk Dullmann. Petabyte databases. *SIGMOD Record*, 28(2):506, 1999.

[3] D. Shasha and P. Bonnet. *Database Tuning: Principles, Experiments, and Troubleshooting Techniques*. Morgan Kaufmann, Singapore, 2002.

[4] Transaction Processing Performance Council. http://www.tpc.org/.

[5] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *OSDI*, 2004.

[6] D. DeWitt and J. Gray. Parallel Database Systems: The Future of High Performance Database Systems. *Communications ACM*, 35(6):85–98, 1992.

[7] Vladimir Kotlyar, Keshav Pingali, and Paul Stodghill. A relational approach to the compilation of sparse matrix programs. In *Proceedings of the Third International Euro-Par Conference on Parallel Processing, Euro-Par97*, pages 318–327, London, UK, 1997. Springer-Verlag.

[8] J. Gray, D.T. Liu, M. Nieto-Santisteban, A.S. Szalay, D. DeWitt, and G. Heber. Scientific Data Management in the Coming Decade. Technical Report MSR-TR-2005-10, Microsoft, Berkeley, Johns Hopkins University, Wisconsin, Cornell, 2005.

[9] R.K. Rew, G.P. Davis, S. Emmerson, and H. Davies. *NetCDF User's Guide for C, An Interface for Data Access, Version 3*. Unidata, University Corporation for Atmospheric Research, Boulder, CO, USA, 1997.

[10] D. Maier and B. Vance. A Call to Order. In *Proceedings of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1–16. ACM Press, 1993.

[11] NCITS H2. Information Technology – Database Languages – SQL. Standard ISO/IEC 9075-XX:1999, ISO, 1999.

[12] L. Libkin, R. Machlin, and L. Wong. A Query Language for Multidimensional Arrays: Design, Implementation, and Optimization Techniques. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 228–239. ACM Press, June 1996.

[13] A.P. Marathe and K. Salem. A Language For Manipulating Arrays. In *Proceedings of the 23rd VLDB Conference*, pages 46–55, 1997.

[14] A.P. Marathe and K. Salem. Query Processing Techniques for Arrays. *The VLDB Journal*, 11(1):68–91, 2002.

[15] P. Baumann. A Database Array Algebra for Spatio-Temporal Data and Beyond. In *Next Generation Information Technologies and Systems*, pages 76–93, 1999.

[16] P. Furtado and P. Baumann. Storage of Multidimensional Arrays based on Arbitrary Tiling. In *Proceedings of the 15th International Conference on Data Engineering, ICDE99*, pages 408–489, March 1999.

[17] S. Sarawagi and M. Stonebraker. Efficient Organization of Large Multidimensional Arrays. In *Proceedigs of the 10th International Conference on Data Engineering, ICDE94*, pages 328–336. IEEE Computer Society Technical Committee on Data Engineering, 1994.

[18] Michael Stonebraker and Greg Kemnitz. The POSTGRES next generation database management system. *Communications of the ACM*, 34(10):78–92, 1991.

[19] Michael Stonebraker and Dorothy Moore. *Object Relational DBMSs: The Next Great Wave*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[20] Vishu Krishnamurthy, Sandeepan Banerjee, and Anil Nori. Bringing object-relational technology to the mainstream. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, SIGMOD99*, pages 513–514, New York, NY, USA, 1999. ACM Press.

[21] Peter A. Boncz, Annita N. Wilschut, and Martin L. Kersten. Flattening an Object Algebra to Provide Performance. In *Proceedings of the Fourteenth International Conference on Data Engineering, ICDE98*, pages 568–577, Washington, DC, USA, 1998. IEEE Computer Society.

[22] Mary E.S. Loomis. Integrating Objects with Relational Technology. *Journal of Object-Oriented Programming Focus On ODBMS*, Jul./Aug.:39, 1992.

[23] David Maier. Making database systems fast enough for CAD applications. *Object-Oriented Concepts, Databases, and Applications*, pages 573–582, 1989.

[24] W3C. Extensible Markup Language (XML) 1.1. Recommendation, http://www.w3.org/TR/xml11/, 2004.

[25] W3C. XML Query (XQuery). Recommendation, http://www.w3.org/TR/xquery/, 2007.

[26] Microsoft. Microsoft support for XML. http://msdn.microsoft.com/sqlxml.

[27] IBM. DB2 XML Extender. http://www.ibm.com /software /data /db2 /extenders /xmlext /library.html.

[28] University of Konstanz, University of Twente, and CWI. MonetDB/XQuery. http://monetdb.cwi.nl/XQuery.

[29] P. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner. Pathfinder/MonetDB: XQuery - The Relational Way. In *Proceedings of the 31st International Conference on Very Large Databases (VLDB 2005)*, 2005.

[30] P.A. Boncz, T. Grust, S. Manegold, J. Rittinger, and J. Teuber. Pathfinder: relational XQuery over multi-gigabyte XML inputs in interactive time. Technical Report INS-E0503, CWI, 2005.

[31] M.M. Astrahan, M.W. Blasgen, D.D. Chamberlin, K.P. Eswaran, J.N. Gray, P.P. Griffiths, W.F. King, R.A. Lorie, P.R. McJones, J.W. Mehl, G.R. Putzolu, I.L. Traiger, B.W. Wade, and V. Watson. System R: Relational Approach to Database Management. *ACM Transactions on Database Systems (TODS)*, 1(2):97–137, 1976.

[32] P.A. Boncz. *Monet : A Next-Generation DBMS Kernel For Query-Intensive Applications*. PhD thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, 2002.