The Papyrus Integrated Data Server

Tim Connors, Waqar Hasan, Curt Kolovson,
Marie-Anne Neimat, Donovan Schneider, Kevin Wilkinson

Hewlett-Packard Laboratories

Abstract

Papyrus is a set of modules and services that enables the parallelization and integration of specialized data managers into one execution environment. Papyrus programs can be transparently targeted to different hardware configurations with each execution strategy optimized for a specific hardware configuration. A Papyrus System consists of a number of clients interfacing to a Papyrus Server. The Server consists of a number of cooperating data managers executing on a multiprocessor system.

1 Motivation

For the last two decades, the database community has been pushing a one engine solution to its customers. This position is reinforced by the acceptance of SQL as the standard DBMS interface language and of the relational server as the standard DBMS engine. The one engine is expected to serve all data-centered applications regardless of the data types managed, the way in which they are used, the performance requirements and the target hardware configuration. This position is as credible as having Detroit announce that one engine has been designed for all terrains and all vehicles, be they trucks, buses or race cars. In the meantime, many application developers have found that the one engine is no panacea, and have built their own specialized data managers. For example, CAD systems, with their need for complex object support, cooperative transactions and high performance, seldom use standard DBMSs for data management. They, as many other applications, have opted for specialization. Specialization provides enhanced specific functionality and improved performance, but it does so at the expense of multi-functionality.

Sophisticated applications often require the management of numerous and varied data types. For example, the need for multiple specialized data managers arises in multimedia systems that must manage text and voice. Similarly, Geographic Information Systems must manage map and attribute data. In such cases, the interoperability of the data managers must be handled in the application. This can be difficult if, for example, the various data managers use different transaction management conventions. Integration can also result in performance degradation because of the data passing and the coordination that must take place in the application. Thus, while the specialization of data managers is driven by functionality and improved performance, their subsequent integration can be difficult and can result in performance degradation.

We focus on the performance of integrated specialized data managers. In particular, we focus on the customization of parallel executions of data manager operators in a variety of computer configurations. This is done by specifying the glue that connects data manager operators in a way that is independent of the computer configuration; and then providing the ability to transparently target the execution to a variety of computer configurations ranging from centralized to distributed environments wherein parallelism and distributed processing are exploited. Parallelization of data manager operators built using Papyrus services is a challenging problem, but a more challenging problem is the parallelization of data manager operators that are built independently of Papyrus and are therefore black boxes to Papyrus.

2 Papyrus Approach

Papyrus is a set of modules and services that enables the parallelization and integration of specialized data managers. Papyrus programs can be efficiently targeted to different hardware configurations and can dynamically adjust at runtime to the number of available resources. A Papyrus System consists of several clients interfacing to a Papyrus Server. The Server consists of several integrated data managers executing on a multiprocessor system. Following are some of the salient aspects of our research.

2.1 Specialization of Data Managers

We address the problem of efficiently storing and retrieving a large variety of data types. Rather than assume that a single system will handle all data types that will ever be needed, we posit that specialization is a more practical and efficient approach. This is similar to EXODUS's approach to customization [4]. Unlike extensible DBMS projects [15, 18] that have used the single extended engine approach, we assume that the management of data is handled by a family of *data managers* where each manager is specialized in handling one or more related data types efficiently. We also assume that data types are accompanied by operators to manipulate them. These operators are basically the interface to the different data managers.

We provide a Storage Infrastructure and a Transaction Manager that may be used to construct specialized data managers. The *Storage Infrastructure* [5, 13] provides persistence and is sufficiently flexible to be the storage foundation for a large number of different data types. It provides data manipulation primitives that can be used to implement various operators and offers optional and automatic logging and recovery. The *Transaction Manager* [14] may be used by operators to coordinate access to shared, recoverable data. Like GENESIS [3], Papyrus offers a few concurrency control algorithms, but it also allows data managers to implement their own.

2.2 Interoperability of Data Managers

It is essential that Papyrus programs be able to span data managers. While heterogeneous database projects [1, 17] concentrate on the integration of heterogeneous data models, schemas and semantics constrained by the autonomy requirement, Papyrus addresses the integration of heterogeneous data types and their operators without such a constraint. Some data managers may be constructed using the Papyrus Storage Infrastructure and Transaction Manager, others may not. The main requirement is that they properly interact with the Papyrus Transaction Manager. Papyrus ensures interoperability among data managers through its Interface Language and Transaction Manager.

The *Papyrus Interface Language* (PIL) is a common interface to all Papyrus-registered data managers [12]. It allows Papyrus clients to transparently invoke more than one data manager. It also permits global optimizations across data managers. Its primary purpose is to invoke data managers' operators, provide mechanisms for processing and communicating data among operators, and to permit optimization and parallelization. It is a functional language with built-in, optimizable constructs for handling data sets. It bears some similarity with FAD [2] in its goals of providing a parallelizable language. While FAD placed much emphasis on data semantics, PIL places more emphasis on the characteristics of data manipulation algorithms through operator properties (see Section 2.3).

The Papyrus Transaction Manager contributes to data manager interoperability. While data managers may support different concurrency control algorithms and client transactions may span data managers, the Transaction Manager serializes transactions over all data managers by using serialization rules supplied by each data manager [14]. Data managers that use non-Papyrus transaction managers must export their serialization rules to the Papyrus Transaction Manager.

2.3 Optimization for Parallel/Distributed Execution

Parallelism can play a significant role in achieving high-performance for data-intensive applications [7]. It is thus imperative that data managers be able to exploit the parallelism and distributed processing power available in different hardware configurations. However, this is difficult because: implementing parallel algorithms is tedious and error-prone; hardware configurations are often replaced by new configurations and can thus invalidate some carefully tuned algorithms; effective resources are sensitive to workload and hence hard to predict; and parallelizing individual operators in isolation is futile because the applicability of parallelism greatly depends on the context in which they are used. The Papyrus Optimizer and Execution Engine optimize and fine-tune programs to adapt easily and effectively to various hardware configurations where parallelism can be exploited.

The *Papyrus Optimizer* compiles expressions containing data managers' operators. Central to its contribution is the transformation of sequential programs into parallel programs. Operators are black boxes to the Optimizer. However, operator implementors are required to abstract the properties of their algorithms so that the Optimizer can optimize their usage. To this end, a number of *properties* may be attached to operators. Examples of such properties are whether an operator has *side-effects* or is *clonable*, i.e., whether it may be cloned to process partitions of the input in parallel. An operator may also be declared to match a template whose behavior is well-understood by Papyrus. Examples of such templates are *draw* an element from a bag, *scan* a bag, or apply a *filter* to an atom. Through these properties and templates, the Optimizer gains a deeper understanding of operators than through a pure rule-based approach [8, 10].

Three forms of parallelism are detected by the Optimizer: *pipelining*, *cloning* and *independent parallelism*, whereby independent sub-expressions are executed in parallel. A formal model of pipelining and strategies for implementing it are developed in [11]. Cost functions are associated with operators. The cost of operators that use the Papyrus Storage Infrastructure is better understood than that of operators that do not use it, and can result in more efficient execution strategies.

2.4 Runtime Support for Parallel/Distributed Execution

The *Papyrus Execution Engine* [6] supports efficient interaction among data managers and their operators regardless of the internals of the operator and of the hardware and processor configuration. Operators are only understood through their interface and properties. Some may be implemented as iterators[1] and some may not. A program may execute in several address spaces in the same machine or on different machines. Some of these machines may be multiprocessors. One of the challenges we have imposed on the Execution Engine is that it be efficient on both single-processor and multiprocessor architectures. The Execution Engine shares some of the goals of Volcano [9] but is more general, in that Volcano only supports operators that are implemented as iterators. Furthermore, the Execution Engine supports some advanced constructs, such as conditionals, which Volcano does not support.

Parallelism depends on many factors that are only known at runtime. For example, the effective number

[1] Iterator functions perform only a partial computation for each invocation of the function. Traditionally, at least three entry points are defined for each function, *open*, *next* and *close*.

of processors available at any time is dependent on the workload. The Execution Engine automates much of the cloning of operators, data partitioning and data passing between operators, to adjust the level of parallelism to the runtime environment.

Parallel and distributed query processing is not limited to the server, as client workstations are assumed to have sufficient processing power to reduce contention on the server and thus contribute to overall improvements in performance. [19] addresses one aspect of client participation in query processing, namely the maintenance of a consistent cache on client workstations.

2.5 Contributions

Thus, Papyrus's contributions are to: provide a Storage Infrastructure and Transaction Manager that can be used to construct specialized data managers; provide an optimizable programming language with a rich computational model to integrate arbitrary data types and their operators; facilitate the parallelization of Papyrus-registered operators; optimize Papyrus programs for different hardware configurations; and efficiently integrate data managers in one execution environment. The Papyrus modules may be used to construct specialized DBMSs. They may also be used independently of one another to construct special-purpose applications.

3 Project Status

Different components of Papyrus are in various stages of design or implementation. A prototype is planned for August 1991. This section describes the status of the project we expect at that time.

The implementation of the Storage Infrastructure will be complete but will not be integrated with the Transaction Manager. The design of PIL will be complete. An initial implementation of the Execution Engine will run on a single-processor, single-computer system, and will execute in a multi-threaded mode to simulate a multiprocessor environment.

The goal of the prototype is to demonstrate the capability of the Papyrus approach of meeting the needs of a data-intensive application that requires the management of non-traditional data types. A Geographic Information System (GIS) will be the demonstration vehicle. GISs typically require the management of vector data, raster image data and attribute data. Many of the algorithms are either CPU intensive or require scanning large amounts of data and are therefore good candidates for parallelism. Current GISs typically relegate the management of attribute data to a relational DBMS and the raster and vector data to software developed specifically for that purpose [16]. For queries that span several kinds of data, either the user or a software application layer must interface to the different systems. We believe that GIS requirements represent a good test of the Papyrus functionality.

References

[1] R. Ahmed, P. DeSmedt, W. Kent, M. Ketabchi, W. Litwin, A. Rafii, M.-C. Shan. Pegasus: A System for Seamless Integration of Heterogeneous Information Sources. *IEEE COMPCON*, February 1991.

[2] F. Bancilhon, T. Briggs, S. Khoshafian, P. Valduriez. FAD, a Powerful and Simple Database Language. *VLDB*, Brighton, England, 1987.

[3] D. Batory, J. Barnett, J. Garza, K. Smith, K. Tsukuda, B. Twichell, T. Wise. *GENESIS: A Reconfigurable Database Management System.* Technical Report #TR-86-07, University of Texas, Austin, March 1986.

[4] M. Carey, D. DeWitt, G. Graefe, D. Haight, J. Richardson, D. Schuh, E. Shekita, S. Vandenberg. *The EXODUS extensible DBMS project: an overview.* CS Technical Report #808, University of Wisconsin-Madison, November 1988.

[5] T. Connors. *The Papyrus Object Library.* Technical Memo HPL-DTD-91-3, January 14, 1991.

[6] T. Connors and D. Schneider. *The Papyrus Execution Engine.* Working Paper.

[7] D. DeWitt, S. Ghandeharizadeh, D. Schneider, A. Bricker, H. Hsiao, R. Rasmussen. The Gamma Database Machine Project. *IEEE Trans. on Knowledge and Data Engineering*, March 1990.

[8] G. Graefe and D. DeWitt. The EXODUS Optimizer generator. *ACM SIGMOD*, May 1987.

[9] G. Graefe. Encapsulation of Parallelism in the Volcano Query Processing System. *ACM SIGMOD*, May 1990.

[10] L. Haas, J. Freytag, G. Lohman, H. Pirahesh. Extensible Query Processing in Starburst. *ACM SIGMOD*, June 1989.

[11] W. Hasan and S. Chaudhuri. *Pipelined Execution in an Extensible DBMS.* Technical Memo HPL-DTD-91-10, February 22, 1991.

[12] W. Hasan. *L1.* Working Paper.

[13] C. Kolovson, C. Nguyen, M.-A. Neimat. *The Papyrus Record Manager.* Working Paper.

[14] M.-A. Neimat and K. Wilkinson. Extensible Transaction Management in Papyrus. *HICCS*, January 1990.

[15] P. Schwarz, W. Chang, J.C. Freytag, G. Lohman, J. McPherson, C. Mohan, H. Pirahesh. Extensibility in the Starburst Database System. *Proc. 1986 Int. Workshop on Object-Oriented Database Systems*, Asilomar, September 1986.

[16] C. Shaffer, H. Samet, R. Nelson. *QUILT: A Geographic Information System based on Quadtrees.* CS Technical Report TR-1885.1, University of Maryland, College Park, MD, July 1987.

[17] A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, vol. 22, no. 3, September 1990.

[18] M. Stonebraker and L. Rowe. The Design of POSTGRES. *ACM SIGMOD*, June 1986.

[19] K. Wilkinson and M.-A. Neimat. Maintaining Consistency of Client-Cached Data. *VLDB*, August 1990.