Single Logical View over Enterprise-Wide Distributed Databases

by
Andrew E. Wade
Objectivity, Inc.

SIGMOD '93

ABSTRACT

Two trends in today's corporate world demand distribution: downsizing from centralized mainframe single database environments; and wider integration, connecting finance, engineering, manufacturing information systems for enterprise-wide modeling and operations optimization. The resulting environment consists of multiple databases, at the group level, department level, and corporate level, with the need to manage dependencies among data in all of them. The solution is full distribution, providing a single logical view to objects anywhere, from anywhere. Users see a logical model of objects connected to objects, with atomic transactions and propagating methods, even if composite objects are split among multiple databases, each under separate administrative control, on multiple, heterogeneous platforms, operating systems, and network protocols. Support for production environments includes multiple schemas, which may be shared among databases, private, or encrypted, dynamic addition of schemas, and schema evolution. Finally, the logical view must remain valid, and applications must continue to work, as the mapping to the physical environment changes, moving objects and databases to new platforms.

I. A New Class of Database Needs

Objectivity Inc. was founded in 1988 to solve database management needs that were not being met by relational or other traditional database management systems (DBMSs). Our unique focus–to solve the production needs of applications in complex distributed environments–has defined our company mission and all aspects of our product design, including architecture, features, and future directions, and has led to more than 50,000 users in production.

The product suite is designed for application developers who want to use object technology to manage complex and highly interconnected data. Typical applications that have these characteristics are multimedia, engineering, software development, telecommunications, scientific, manufacturing, transportation, publishing, and financial applications. These applications also typically involve distributed, multi-database, multi-platform, heterogeneous computing environments.

The Objectivity product suite consists of Objectivity/DB (a high performance and flexible object database management system (ODBMS)), and a broad range of development tools and services to help developers move applications quickly from prototype to full production using object information management. The Objectivity/DB engine provides concurrent, high-performance access to complex and highly interconnected objects distributed over multiple databases on multiple heterogeneous machines. The tools and services provided by Objectivity are delivered as a combination of direct products, third-party relationships, and integration services. These tools and services include ad hoc query and report , legacy application migration , legacy database connectivity, database administration , object analysis and design, integration with third-party CASE environments, class libraries, and GUI and application builders. All Objectivity products are designed to be compatible with and complementary to other technologies, including traditional DBMSs.

Developers typically customize their environment using many related products, such as programming languages, compilers, design tools, CASE environments, and code debuggers. Objectivity has a complementary strategy, based on standards for compatibility with the broadest range of existing products, and on strategic alliances with vendors of such products and major users. This provides developers with the widest choice of solutions, and also allows tight integration and vendor cooperation for highly productive development environments.

II. Target Applications

Objectivity/DB technology has significant benefits for applications that share the following characteristics:

- Object-oriented (C++)
- Objects with complex structure or with many relationships (C/C++)
- Objects or data that are distributed across multiple machines or databases (C/C++)

Major portions of the software industry are adopting object technology in order to improve productivity through better re-use of proven software and easier extensibility of systems. An ODBMS is an important component of this environment because it enables object applications to communicate to the DBMS using objects. This saves the overhead of translating from objects to the records or tuples required by RDBMSs and other DBMS technologies.

For complex applications, those that involve complex data structures, complex operations on those structures, and high performance for those operations, traditional DBMSs are generally cumbersome and too slow by orders of magnitude. RDBMS-based applications that involve traversal of highly interconnected data generally find themselves bogged down in DBMS operations such as joins that degrade performance significantly.

Objectivity/DB provides a powerful mechanism to support not only complex objects with arbitrary structures and multiple varying-sized elements, but also high-speed direct inter-object references or associations. Composite objects can be built of any number of other objects, to any number of levels of depth, and any number of levels of breadth, with automatic propagation of operations.

As applications are moved from host-based systems to distributed computing environments, data distribution and application requirements can change significantly.

Whether data is centralized or distributed, applications need a DBMS that supports transparent access to objects from all the workstations. This full distribution capability, with access to objects on any platform from all other platforms, is a unique strength of Objectivity/DB. Heterogeneous mixtures of platforms are an integral feature of distributed computing environments because users want to choose the best combination of price and

SIGMOD /5/93/Washington, DC,USA
© 1993 ACM 0-89791-592-5/93/0005/0441...$1.50

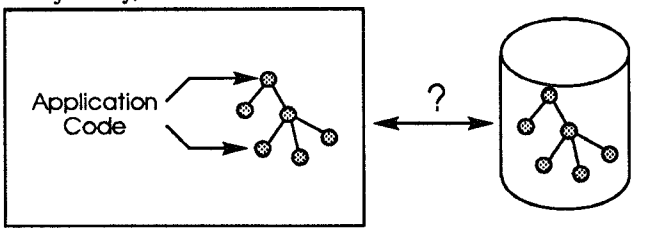
performance, regardless of the vendor. As a result, as applications interconnect data across departments and enterprises, the ability to transparently access objects across multiple databases throughout the distributed environment is critical.
Multiple databases may be under control of separate organizations, while composite objects span those databases. Applications should see only a logical view of objects connected to objects, with the ODBMS managing logistics such as data format translation, moving objects from remote disk into local cache, atomic transactions across databases, and propagation of methods.
Applications with these requirements are found in many areas, and many are now migrating to object technology. Objectivity has experience with: Multimedia; CASE; Configuration Management; Telecommunications; Document Creation and Management; Transportation Design and Control; Manufacturing Design; Measurement, Control, and Automation; Office Automation and GroupWare; Publishing; Service Management and Support; Economic and Financial Modeling; Scientific Research and Development; Imaging and Visualization; Medical Information; Geographic Information Systems; Network Management Systems; CAD; and CAM.

III. Architectural Alternatives

Objectivity's experience with traditional DBMSs, proprietary high performance DBMSs, traditional applications, and object technology inspired a new architecture for database management. The goal was to use object technology to provide the data integrity of RDBMSs, the flexible data structures of proprietary systems, and the in-memory performance of interactive applications, all in a production quality ODBMS.
The challenge of meeting these requirements dictated a new architecture. Through past experience in applications and DBMSs for production environments, the Objectivity technical team and technical advisory board were familiar with the limitations of previous architectural approaches, including the virtual memory operating system approaches of the 1960s and 1970s, the conventional, centralized host-based DBMS approach of the 1970s, and the client/server architectures of the 1980s. A new architecture was designed to include the advantages of each of these, avoid their disadvantages and limitations, and provide a DBMS that could deliver applications into production.

IV. Limitations of Virtual Memory Architecture

The virtual memory, or memory mapping, approach was originally designed for single program use, not for shared, persistent environments. Many applications whose performance requirements were not satisfied by traditional DBMSs turned to this approach, because of ease of storing and retrieving data structures, and executing with near-native in-memory performance. However, these same application builders ran into problems. Virtual memory systems tended to perform well only when all the data fit into memory. Additional data triggered swapping mechanisms with unacceptable performance. For sustained high performance beyond available memory, a direct database management mechanism was needed to efficiently move objects into cache and to manage multiple databases and addresses beyond the capabilities of 32-bit virtual memory systems (and swap spaces often limited to only 256MB).
Virtual memory systems also tend to have data integrity problems. Allowing applications to directly use pointers to physical locations in memory leads to integrity problems from invalid pointers, which causes crashes or sometimes subtle data corruption. For traditional applications, these integrity problems were contained by application crashes and loss of session information. For a database, however, the problems are much more insidious. As the data is swapped out or de-cached at commit time, more pointers may become invalid, leading to subtle corruption that might not be noticed until much later.Because of these dangers, Objectivity selected the RDBMS approach of providing a safe interface between applications and data as another desirable characteristic for Objectivity/DB.



Single-user Application Process

Direct Pointer Access Can Corrupt Data

Figure 1 Virtual Memory Architecture is not for Shared Persistent Environments

Virtual memory systems also tend to perform poorly when concurrent access is required by users, especially from other machine architectures. The approach of surrendering control of the fundamental memory- and disk-management to the operating system had been used for prototypes and small applications, but large production environments require the ability to control page size, clustering, caching, and swapping algorithms independent of a particular operating system's virtual memory process swapping. Access across platforms and databases requires a mechanism that can transparently and efficiently operate across a fully distributed and heterogeneous computing environment, without being limited by a single virtual memory system implementation. Architectural control at the object level would be required to support access to sparse or inefficiently clustered data; otherwise, virtual memory mechanisms will read in unnecessary amounts of extraneous data. Control at the object level is also required to allow object-level security, to allow the DBMS maximum flexibility in storage management, and to optimize performance. Because of these requirements, Objectivity added an on-demand object address interface as an architecture requirement for Objectivity/DB.

V. Limitations of Central Server Architectures

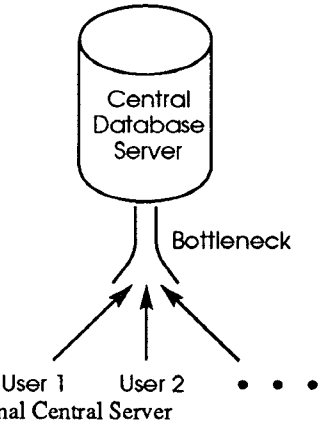


Figure 2 Traditional Central Server

A very different architecture approach is used for the centralized, mainframe-like design of RDBMSs and traditional DBMSs. These systems can, in some cases, manage very large amounts of data via direct DBMS memory- and disk-management. However, they

support only simple structures, and completely lack support for dynamically varying sized structures, arbitrary structures, direct inter-object connections, and composite objects.
These systems also isolate data buffers and database activity to a separate process and often a separate machine. For new distributed computing environments, this requires a slow inter-process interface to access each object. Typically milli-seconds or tens of milli-seconds are required for inter-process communication as compared to intra-process operations, which run in a few cycles or a few tenths of a microsecond on a 10 MIP machine. Also, forcing all access to go through a central server creates a fundamental bottleneck when all users must wait in a queue at the server for all operations. This limits scalability for multiple users. Objectivity recognized that a full distribution of data and processing requires a fundamentally different approach.

VI. The Objectivity/DB Fully-Distributed Architecture

The Objectivity solution was to design a new architecture incorporating the best features of virtual memory and central server architectures. It was possible to provide the arbitrary structure access and near-native performance of virtual memory by mapping objects directly into a cache in the application's address space and executing there. However, to allow the ODBMS to perform memory- and disk-management for high performance with more complex data, more users, multiple databases and distributed users, application access to the objects in this cache is through one level of pointer indirection. The cost for this was shown to be negligible in production environments, as has been verified in recent benchmarks, where the greatest performance bottleneck was disk I/O.
The advantages of this were that the ODBMS could now provide an extra level of integrity, ensuring that applications could never de-reference a "stale" pointer due to commit or swapping, and subtly corrupt the database. The ODBMS could implement direct

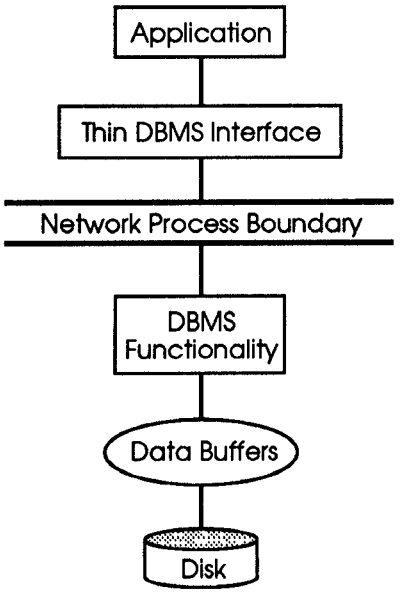


Figure 3 Traditional Client-Server Architecture

management of memory and disk, clustering, caching, multiple buffer management, and replacement algorithms for optimal application performance. The ODBMS could also provide a fully distributed approach, portable across all platforms, not limited by a single operating system's virtual memory limitations or restrictions.
In Figure 3 the traditional database client-server (actually a central server) architecture is shown. Figure 4 shows the Objectivity/DB new, fully-distributed architecture. The network/process boundary in each figure shows the position of the inter-process boundary is significant, because operations that cross this boundary suffer an overhead of 10,000x (ones or tens of milli-seconds IPC versus sub-micro-second cycle time within the process).

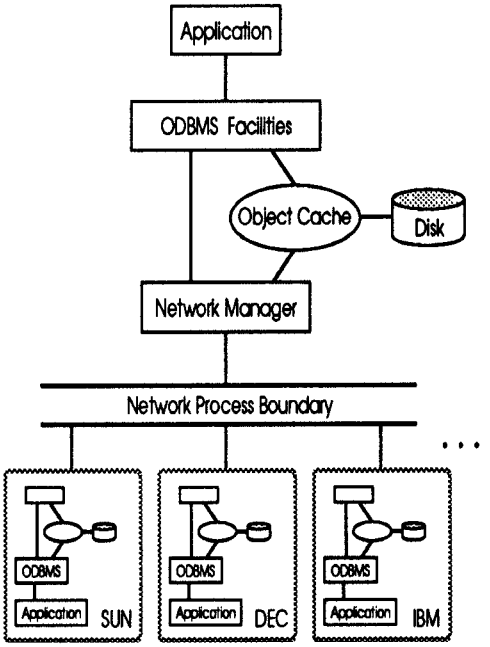


Figure 4 Objectivity/DB Fully Distributed Architecture

In traditional architectures, the application's address space contains a very thin layer that packages a database request. For example a high-level query is sent across the interface to the server where all the buffers are maintained and all the processing occurs. The answer, typically a small number of bytes, is then returned across the interface.
This approach works well when the request and the answer are small and the operation is long, so that network overhead is amortized over a long operation. High-level queries that can take seconds, or applications like automatic teller machines or airline reservation systems that are sequentially accessed by each user can make efficient use of this architecture. However, it works poorly when the user must make repeated database requests, traversing from one object to another, performing complicated operations, all at interactive speeds.
To distribute processing and storage, the Objectivity/DB architecture moves the process interface boundary down, providing an object cache within the application address space. As objects are accessed, they are brought into the cache, and can then be accessed at native memory speeds. Traversal, creation, naming, indexing, method propagation, and other functions are all executed locally. When the application requests an object that is not in the cache, the ODBMS traps this request; the network manager locates the database containing the object, the volume containing the database, obtains read/write locks if necessary, moves the object into the cache, performs heterogeneity transformations if necessary, and sets up the pointer indirection, so that later operations are at native speeds. This is all done transparently for the application.

In this way, virtual memory is used for local operations, while simultaneously providing transparent fully distributed access to multiple databases anywhere on the network of heterogeneous computers. The user may choose to localize objects in a single database or multiple databases, or on a single or multiple central servers. Processing and storage are provided according to the applications and the resources available. The result is full distribution.

- object distribution: objects can be located anywhere, and accessed from all platforms
- control distribution: objects can be distributed among multiple databases, each under separate administrative control
- single logical view regardless of location or format

Databases can be for private use, group use, or department use; for example for electrical design, packaging, test, manufacturing, finance. Each is under the administrative control of its owner, who determines when to back up, shut down, and how to control access. Yet composite objects and all database operations can extend across all databases. All of this occurs transparently. There is no need for the user or application developer to know which database an object is in, on which machine, or to manually move it to a private database or into memory.

The Objectivity/DB architecture transparently handles differences among heterogeneous machines and operating systems, including:

- byte ordering
- floating point formats
- compiler differences
- structure/class padding and alignment

All this is done at the object level, so it is not necessary to process more objects than actually requested. Further, this unique mechanism allows use of the exact binary representation of the native environment. Unlike approaches that change the representation by maximal padding for the worst-case architecture, this saves space and allows compatibility with pre-existing binary libraries so a structure can be passed directly to an operating system routine.

The Objectivity/DB architecture also exploits distributed computing environment resources efficiently, for improved performance as networks grow. With traditional architectures, when additional capacity or performance is required, the only choice is to replace or upgrade the server. In this fully distributed architecture, a new machine adds its own processing resources to the network and can automatically access databases on other machines without reconfiguring a server. Of course databases may also be moved to the new machine to take advantage of additional storage, or to reduce network traffic. This allows maximum use of resources and provides the ability to take advantage of price and performance improvements in platforms while preserving the current hardware investment and not requiring changes to applications.

Objectivity/DB provides for a federation of databases and maintains a catalog of known databases and database schemas. Distributed schemas are managed by the federation, and databases can share schemas or have their own object definitions. Shared schemas reduce storage requirements and simplify schema evolution compared to systems that store a copy of the schema in each database. Cross-database object references are also managed by the federation for full referential integrity. Schemas and databases may be added, removed, secured, temporarily detached from or reattached to the federation with flexible administrative control. The Objectivity/DB architecture provides rich features for managing a distributed environment of multiple applications, databases, platforms.

VII. Single Logical View

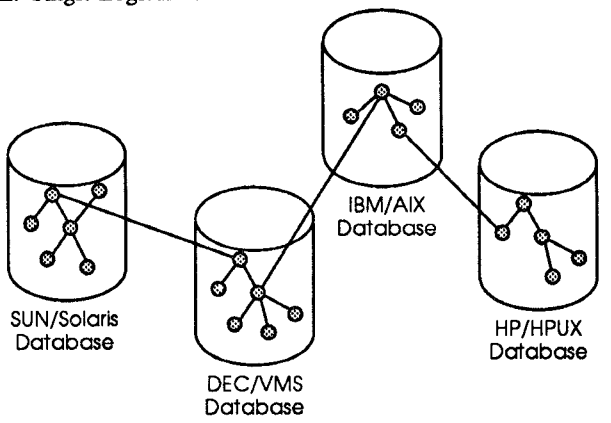


Figure 5 Single Logical View of Composite Object

A logical view is maintained independent of the physical view, without exposing the developer to the physical structure of the database. This allows developers to work solely at the logical level, with objects and versions, arbitrarily complex objects (arbitrary number of varrays in objects), and arbitrary composite objects (arbitrary number of levels of depth, arbitrary number of composites threading through single object). Independently, these logical structures can be mapped to the storage hierarchy of federated databases, multiple databases, multiple containers within databases, and objects, allowing localization for administrative control, performance, and security. All database operations transparently respect this logical-to-physical mapping, including, for example, atomicity of long and short transactions, recovery, and backup. Multiple schemas allow users to share schemas across databases, but also to create their own private, perhaps encrypted, schemas, without any central coordination. Through a gateway (e.g., with a partner, Persistence, Inc.), this logical view has begun to extend, also, to relational DBMSs such as Oracle and Sybase, and via surrogate object methods, users have even extended it to include proprietary, flat-file formats, albeit with some limitations on granularity and transactions.

VIII. Application and Conclusion

The success of new technology depends not only on its fitness to solve a problem, but also on its adaptability to the user's environment. As users begin to adopt object technology and use complex interconnected objects, they need their ODBMS solution to work in their existing environment: it must interconnect to pre-existing systems, file formats, and databases, and it must work in the heterogeneous environment of hardware, networks, operating systems. When a departmental database (e.g., human resources, sales order entry, manufacturing, engineering) wants to connect its information model with others, it must allow transparent cross-database objects and method propagation, but still allow each department to control its own database administratively, and create its own schemas. This is how the single logical view allows users to gradually adopt object technology and incrementally migrate towards full enterprise integration.