DataBlitz Storage Manager: Main-Memory Database Performance for Critical Applications

J. Baulier P. Bohannon S. Gogate C. Gupta S. Haldar S. Joshi A. Khivesera
H. Korth P. Mcilroy J. Miller P. P. S. Narayan M. Nemeth R. Rastogi
S. Seshadri A. Silberschatz S. Sudarshan* M. Wilder C. Wei
Bell Laboratories, Murray Hill

1 Introduction

General-purpose commercial disk-based database systems, though widely employed in practice, have failed to meet the performance requirements of applications requiring short, predictable response times, and extremely high throughput rates. Main memory is the only technology capable of these characteristics.

DataBlitz[1] is a main-memory storage manager product that supports the development of *high-performance* and *fault-resilient* applications requiring concurrent access to shared data. In DataBlitz, core algorithms for concurrency, recovery, index management and space management are optimized for the case that data is memory resident.

2 DataBlitz Architecture and Features

In this section, we give a high-level overview of the architecture and features of the DataBlitz Storage Manager product implemented at Bell Laboratories (for more details, see [1]).

Direct Access to Data. DataBlitz is designed to allow direct access to data in shared memory, without overhead for interprocess communication. Applications map the entire database as well as logging and locking information into their virtual address space (see Figure 1). Thus, unlike disk-based databases, accesses in DataBlitz

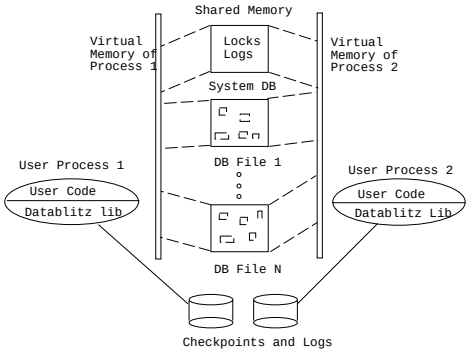

Figure 1: Architecture of the DataBlitz System

do not incur network latency or buffer manager overhead. This is an important reason for the superior performance of DataBlitz over commercial disk-based systems.

Configurable, Multi-level API. Data organization is supported in DataBlitz with persistent space management, and concurrent index management based on both hashing and tree structures. An important principle that has guided the implementation of DataBlitz has been a toolkit approach and support for multiple interface levels. The former implies, for example, that logging facilities can be turned off for data that need not be persistent, and locking can be turned off if data is private to a process. The latter principle means that low-level index components are exposed to the user so that critical application components can be optimized with special implementations. Although application developers are likely to prefer the high-level relational interface for its ease of use, our experiments indicate that substantial performance gains can be realized by using the low-level interfaces. For instance, for a simple phone number lookup application, DataBlitz outperformed a commercial disk-based database system by almost a factor of 40! Also,

*Current address: Department of Computer Science, IIT Bombay.

[1] A commercially available production software platform which has evolved from the *Dali* research project at Bell Laboratories. (See http://www.bell-labs.com/project/dali/).

exporting the low-level interfaces enables DataBlitz to be customized to meet specific application needs such as a small *footprint* (that is, executable size).

Resilience to Failures. A key design requirement for DataBlitz has been resilience to software and hardware faults. To this end, features to protect data from corruption (due to stray application pointers) using *codewords*, and to recover synchronization abstractions (latches) from process failure have been developed. In addition, tools for auditing internal data structures, space usage and database structures like indices are currently being developed. Further, an advanced multi-level transaction model designed for main-memory provides for highly concurrent index access while allowing recovery from system halting failures. Also, applications can avail of the *hot standby* feature which provides high reliability and availability via redundancy on a secondary machine. This is a crucial requirement in telecom and other mission-critical applications for whom overall downtime (including the time spent recovering from failures) must be no more than a few minutes per year. Other features being driven by such applications (that will be supported in the future) include *on-line schema evolution* and *on-line software upgrades.*

Scalable and Performance-driven Architecture. In order to ensure predictability of response times for user applications and their scalability in *symmetric multiprocessor environments*, DataBlitz provides support for fine-grained concurrency control at both the lock (e.g., record level locking) and latch levels (e.g., for protecting system structures like the lock table), and *fuzzy* checkpoints that minimally interfere with transaction processing. In the absence of disk I/O, the performance of a database query is governed by the number of CPU instructions necessary to execute it. To minimize CPU cycles, DataBlitz relies heavily on caching plans and other structures (e.g., iterators, cursors) between consecutive queries. In a main-memory environment, since locking and latching overheads constitute almost 50% of the execution time for a simple lookup, special attention has been paid in DataBlitz to reducing their cost. Latches are implemented in user space as spin locks (since system semaphores can be very expensive), while locking performance is optimized by statically allocating lock structures as opposed to conventional implementations based on a dynamic lock table.

Minimizing Storage Overhead. In order to reduce storage space overhead and the path length for data accesses, and since the database resides entirely in main-memory, DataBlitz does not employ a traditional storage architecture that is based on slotted pages. Instead, free lists keep track of free space for the entire database, and direct physical pointers to objects are permitted. Note that direct physical pointers improve the speed of access for objects at the cost of complicating the task of relocating/compacting them. Further space savings are achieved by not storing keys in indices; instead, indices only store pointers to objects and the objects themselves are used to extract key information when the index is traversed.

3 Applications

Applications in financial trading, electronic commerce and real-time billing are among those well-suited for DataBlitz's features, as well as traditional telecommunications applications like switching and call routing.

Redesigning Special-Purpose Systems. For several decades, the practice in main-memory systems has been to code special-purpose, custom systems. This practice mainly arises from the lack (until recently) of viable commercial alternatives. Recently, we have seen increasing interest in employing general-purpose storage managers in place of existing legacy systems. Examples of these systems include military systems and switching in telecommunications. In the telecommunications environment, transactions are mostly read-only and involve looking up switch configuration information (connections between incoming and outgoing trunks), routing tables and information related to services subscribed by users (e.g., call waiting, call forwarding). The lookups require sub-millisecond response times since the switching and routing functions are performed during call setup whose duration is typically a few tens of milliseconds.

Re-Inventing Traditional Applications. Traditional batch systems are continually subject to reinvention as on-line processes. For instance, real-time event aggregation is used to turn formerly batch-based applications such as telecommunication billing into real-time applications. This capability is the key enabler of pre-paid accounts, where it must be determined as part of the set-up of a call (whose duration is a few milliseconds) whether the account has sufficient funds for the call, and if so, for how long the call can be allowed to continue. Another application enabled by real-time event aggregation is fraud prevention.

References

[1] P. Bohannon, D. Lieuwen, R. Rastogi, S. Seshadri, A. Silberschatz, and S. Sudarshan. The architecture of the Dalí main-memory storage manager. *Multimedia Tools and Applications*, 1997.