Challenges in Embedded Database System Administration

Margo I. Seltzer, Harvard University
Michael A. Olson, Sleepycat Software

Database configuration and maintenance have historically been complex tasks, often requiring expert knowledge of database design and application behavior. In an embedded environment, it is not possible to require such expertise or to perform ongoing database maintenance. This paper discusses the database administration challenges posed by embedded systems and describes how Sleepycat Software's Berkeley DB architecture addresses these challenges.

1 Introduction

Embedded systems provide a combination of opportunities and challenges in application and system configuration and management. As an embedded system is most often dedicated to a single application or small set of cooperating tasks, the operating conditions of the system are typically better understood than those of general-purpose computing environments. Similarly, as embedded systems are dedicated to a small set of tasks, one expects that the software to manage them would be small and simple. On the other hand, once an embedded system is deployed, it must continue to function without interruption and without administrator intervention.

Database administration has two components, initial configuration and ongoing maintenance. Initial configuration includes database design, manifestation, and tuning. The instantiation of the design includes decomposing the data into tables, relations, or objects and designating proper indices and their implementations (e.g., B-trees, hash tables, etc.). Tuning the design requires selecting a location for the log and data files, selecting appropriate database page sizes, specifying the size of in-memory caches, and determining the practical limits of multi-threading and concurrency. As we expect that an embedded system is created by experienced and knowledgeable engineers, requiring expertise during the initial system configuration process is acceptable, and we focus our efforts on the ongoing maintenance of the system. In this way, our emphasis differs from other projects that focus on automating database design, such as Microsoft's AutoAdmin project [3]and the "no-knobs" administration that is identified as an area of important future research by the Asilomar authors [1].

In this paper, we focus on what the authors of the Asilomar report call "gizmo" databases [1], databases that reside in devices such as smart cards, toasters, or telephones. The key characteristics of these databases are that their functionality must be completely transparent to users, no explicit database operations or database maintenance is ever performed, the database may crash at any time and must recover instantly, the device may undergo a hard reset at any time (requiring that the database return to its initial state), and the semantic integrity of the database must be maintained at all times. In Section 2, we provide more detail on the sorts of tasks typically performed by database administrators (DBAs) that must be automated in an embedded system.

The rest of this paper is structured as follows. In Section 2, we outline the requirements for embedded database support. In Section 3, we discuss how Berkeley DB is conducive to the hands-off management required in embedded systems. In Section 4, we discuss novel features that enhance Berkeley DB's suitability for the embedded applications. In Section 5, we discuss issues of footprint size. In Section 6, we discuss related work, and we conclude in Section 7.

2 Embedded Database Requirements

Historically, much of the commercial database industry has been driven by the requirements of high performance online transaction processing (OLTP), complex query processing, and the industry standard benchmarks that have emerged (e.g., TPC-C [9], TPC-D [10]) to allow for system comparisons. As embedded systems typically perform fairly simple queries and only rarely require high, sustained transaction rates, such metrics are not nearly as relevant for embedded database systems as are ease of maintenance, robustness, and a small memory and disk footprint. Because of continuously falling hardware prices, robustness and ease of maintenance are the key issues of these three requirements. Fortunately, robustness and ease of maintenance are side effects of simplicity and good design. These, in turn, lead to a small size, contributing to the third requirement of an embedded database system.

2.1 The User Perspective

Users must be able to trust the data stored in their devices and must not need to manually perform any database or system administration in order for their gizmo to perform correctly.

In the embedded database arena, it is the ongoing maintenance tasks that must be automated, not the initial system configuration. There are five tasks traditionally performed by DBAs, which must be performed automatically in embedded database systems. These tasks are log archival and reclamation, backup, data compaction/reorganization, automatic and rapid recovery, and reinitialization from an initial state.

Log archival and backup are tightly coupled. Database backups are part of any recoverable database installation, and log archival is analogous to incremental backup. It is not clear what the implications of backup and archival are for an embedded system. Consumers do not back up their telephones or refrigerators, yet they do (or should) back up their personal computers or personal digital assistants. For the remainder of this paper, we assume that some form of backups are required for gizmo databases (consider manually re-programming a telephone that includes functionality to compare and select long-distance services based on the caller's calling pattern and connection times). Furthermore, these backups must be completely transparent and must not interrupt service, as users should not be aware that their gizmos are being backed up, will not remember to initiate the backups explicitly, and are unwilling to wait for service.

Data compaction or reorganization has traditionally required periodic dumping and restoration of database tables and the recreation of indices in order to bound lookup times and minimize database growth. In an embedded system, compaction and reorganization must happen automatically.

Recovery issues are similar in embedded and traditional environments with two significant exceptions. While a few seconds or even a minute of recovery is acceptable for a large server installation, consumers are unwilling to wait for an appliance to reboot. As with archival, recovery must be nearly instantaneous in an embedded product. Additionally, it is often the case that a system will be completely reinitialized, rather than performing any type of recovery, especially in systems that do not incorporate non-volatile memory. In this case, the embedded database must be restored to its initial state and must not leak any resources. This is not typically an issue for large, traditional database servers.

2.2 The Developer Perspective

In addition to the maintenance-free operation required of embedded database systems, there are a number of requirements based on the constrained resources typically available to the "gizmos" using gizmo databases. These requirements are a small disk and memory footprint, short code-path, programmatic interface (for tight application coupling and to avoid the time and size overhead of interfaces such as SQL and ODBC), support for entirely memory-resident operation (e.g., systems where file systems are unavailable), application configurability and flexibility, and support for multi-threading.

A small footprint and short code-path are self-explanatory; however, what is not as obvious is that the programmatic interface requirement is their logical result. Traditional interfaces, such as ODBC and SQL, add a significant size overhead and frequently add multiple context/thread switches per operation, not to mention several IPC calls. An embedded product is unlikely to require the complex and powerful query processing that SQL enables. Instead, in the embedded space, the ability for an application to obtain quickly its specific data is more important than a general query interface.

As some systems do not provide storage other than memory, it is essential that an embedded database work seamlessly in memory-only environments. Similarly, many embedded operating systems have a single address space architecture, so a fast, multi-threaded database architecture is essential for applications requiring concurrency.

In general, embedded applications run on gizmos whose native operating system support varies tremendously. For example, the embedded OS may or may not support user-level processes or multi-threading. Even if it does, a particular embedded application may or may not need it, e.g., not all applications need more than one thread of control. An embedded database must provide mechanisms to developers without deciding policy. For example, the threading model in an application is a matter of policy, and depends not on the database software, but on the hardware, operating system and library interfaces, and the application's requirements. Therefore, the data manager must provide for the use of multi-threading, but not require it, and must not itself determine what a thread of control looks like.

3 Berkeley DB: A Database for Embedded Systems

The current Berkeley DB package, as distributed by Sleepycat Software, is a descendant of the hash and B-tree access methods that were distributed with the 4BSD releases from the University of California, Berkeley. The original software (usually referred to as DB 1.85), was originally intended as a public domain replacement for the