LGeDBMS: a Small DBMS for Embedded System with Flash Memory

Gye-Jeong Kim Seung-Cheon Baek Hyun-Sook Lee Han-Deok Lee
Moon Jeung Joe

Embedded System Technology Group, Information Technology Lab.
LG Electronics Institute of Technology
16 Woomyeon-Dong, Seocho-Gu, Seoul, Korea
{gjkim81, white413, hslee04, cyber93, joemoon}@lge.com

ABSTRACT

The ever-increasing requirement of high performance and huge capacity memories of emerging consumer electronics appliances, such as mobile phone, digital camera, MP3, PMP, PDA, etc., has led to the widespread adaptation of flash memory as main data storages, respectively. As a result, managing the data on flash memory has been gaining in significant to satisfy the requirement of mobile embedded applications. However, the read/write/erase behaviors of flash memory are radically different than that of magnetic disks which make traditional database technology irrelevant. In this paper, we introduce *LGeDBMS*, a scale-downed DBMS engine designed for flash memory and its application. Finally, we demonstrate a PIM(Personal Information Management) application on a mobile phone using *LGeDBMS*.

1. INTRODUCTION

As mobile embedded systems evolve into data centric and multimedia-oriented applications, storage with high performance and huge capacity has become necessary. For this reason, flash memory has become an indispensable component in mobile embedded systems because of its versatile features such as non-volatility, shock resistance, low cost, small size and high density.

However, the read/write/erase behaviors of flash memory are radically different than that of other programmable memories such as volatile RAM and magnetic disks. Perhaps more importantly, memory blocks in a flash memory can be written to only a limited number of times, between 10,000 and 1,000,000, after which they wear out and become solid state.

In fact, flash memories come in two flavors, NOR and NAND, that are also quite different from each other. In both types, write operations can only clear bits (change their value from 1 to 0). The only way to set bits (change their value from 0 to 1) is to erase an entire region of memory. These regions have fixed size in a given device, typically ranging from several kilobytes to hundreds of kilobytes and are called erase units. NOR flash memory, the older type for code storage, is a random-access device that is directly addressable by the processor. Each bit in a NOR flash memory can be individually cleared once per-erase-cycle of the erase unit containing it. NOR flash memory suffers from high erase times. NAND flash memory, the newer type for data storage, enjoys much faster erase times, but it is not directly addressable, access is by page (a fraction of an erase unit, typically 512 bytes) not by bit or byte, and each page can be modified only a small number of times in each erase cycle. That is, after a few writes to a page, subsequent writes cannot reliably clear additional bits in the page; the entire erase unit must be erased before further modifications of the page are possible.

Because of these peculiarities, storage management techniques that were designed for other types of memory devices, such as magnetic disks, are not always appropriate for flash memory. To address these issues, flash-specific file system techniques have been developed with the widespread introduction of flash memories in the early 1990s[5][1].

Moreover, as the flash memory capacity is doubled in every year[8], managing the data on flash memory has been gaining in significant to satisfy the requirement of mobile embedded applications. Generally, database management helps to separate data management code from application code, thereby simplifying and developing application code easier. However, due to the resource limitations, small memory and low computing power, the traditional DBMS technology can not apply to mobile embedded systems.

To address these problems, we have implemented *LGeDBMS*, a DBMS for mobile systems with flash-memory. *LGeDBMS* has the following features: (1) optimized to flash memory with LFS design principle[7], (2) compact size suitable for consumer electronics appliances, and (3) transaction process based on flash memory characteristics.

Some approaches have been developed for scaling down DBMS [2][9][6][4]. PicoDBMS[2] is a database system designed to execute on resource-constrained smart card. PicoDBMS aims to reduce the total database size by vertically decom-

VLDB '06, September 12-15, 2006, Seoul, Korea.

posing relations in one column partitions to limit the repetition of column values. While such storage model is appropriate for EEPROM memories that can be erased one word at a time, it is inappropriate for flash memory that can be erased only in large blocks, since tuple may be distributed among multiple blocks. Light version of popular DBMS like Sybase Adaptive Server Anywhere[9], Oracle 8i Lite[6] or DB2 Everywhere[4] have been primarily designed for portable computers and PDA. They use relatively much RAM and stable memory and do not address the more serve limitation of mobile phones.

In this paper, we describe *LGeDBMS*, a small DBMS for embedded system with flash memory. Section 2 explains *LGeDBMS* storage system and transaction process specialized for flash memory. Section 3 demonstrates efficiency of PIM component, a middleware for mobile phone applications, on *LGeDBMS* through PC simulators and mobile phones.

2. OVERVIEW OF *LGeDBMS*

LGeDBMS is a relational database management system designed for efficient data management and easy data access in embedded mobile system.

LGeDBMS has the following features:

Table 1: Features of *LGeDBMS*

Function	LGeDBMS
SQL/API	C API
DDL	Create/Drop Table, Index
DML	Insert, Update, Delete
Search	Single Select
Transaction	Begin, Commit, Redo (Atomicity, Durability Transaction)
Object	Table, Index
Data Type	Char, Var Char, Int, Boolean, Float, Double, BLOB, Date
Recovery	Versioning Based Recovery
Memory Management	Fixed Size Cache(Configurable)
Code Schema	UCS2
Storage Media	Disk, NAND/NOR Flash Memory
Supported OS	REX, pSOS, QNX, Windows, Linux, MacOS

2.1 Storage System

LGeDBMS adopts principles of Log-structured file system (LFS) whose I/O unit is a page. LFS is a file system originally developed for disk[7]. It organized a disk as a log which is an ideally continuous medium. In this scheme, data are always written to the end of the log in order to enjoy good write performance due to no seek and no rotational delays. While LFS is not in widespread use on disks, it makes perfect sense on flash memory[3]. On flash memory, old data cannot be overwritten before erasing, so the modified data must be written out-of-place. Therefore, the design principle of LFS is used by most of flash-specific file system.

We apply the design principle of LFS for designing the storage system and the transaction process of *LGeDBMS*. Firstly,

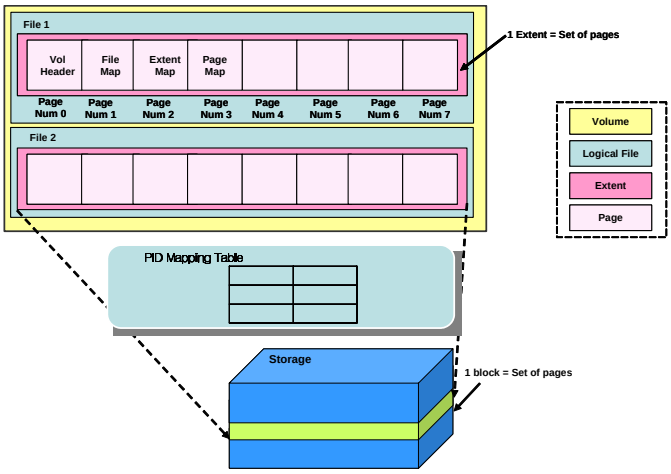

Figure 1: Storage Structures of *LGeDBMS*

we focus on the storage system of *LGeDBMS*. Figure 1 shows the data storage structures of *LGeDBMS*. *LGeDBMS* has hierarchical structures of volumes, logical files, extents, and pages. The structures are managed by a volume header, a file map, an extent map, a page map, and a PID mapping table.

A volume corresponds to a part or whole flash memory for storing data. A volume header occupies one page and contains information as following: (1) volume name, (2) file information such as the total number of logical files in the volume, the number of file maps, the last logical file number, and a position of the first file map, (3) extent information such as the total number of extents in the volume, the number of extent maps, the last extent number, and a position of the extent map, and (4) page information such as the total number of the pages and a position of the page map.

A file map manages logical files such as table files and index files created in a volume. And it contains an extent fill factor and the extent number which includes the first extent map. The extent fill factor is the maximum number of extents that can be contained in a file.

An extent is fixed number pages which are logically continuous. And the number of pages in one extent can be configurable. An extent map contains the next extent number to construct a linked list of extents in the same file.

A page is an I/O unit of *LGeDBMS*. A page map indicates whether each page in an extent is occupied or not. And each extent is represented as a byte unit. Each map is maintained and managed as a linked list.

A PID mapping table has mapping information from logical pages to physical pages. By setting the size of a logical page as the size of a physical page, *LGeDBMS* is able to reduce additional I/Os due to unaligned pages or other layer information such as a file system. For example, when *LGeDBMS* is implemented on a file system, one logical page can be written to two physical pages due to additional file system

information.

Furthermore, a PID mapping table resolves in-place update overhead and wear-leveling issue when *LGeDBMS* directly controls flash memory at the device driver level. (If *LGeDBMS* runs on a file system, the file system resolves those issues.) Without PID mapping table, data should be written in-place after erase, which degrade performance. And frequent erasing of some particular locations of flash memory could quickly deteriorate the overall lifetime of flash memory.

Complex wear-leveling technique consumes relatively much more memory and expensive management cost, which is inadequate to mobile embedded system with resource limitation. Therefore, *LGeDBMS*'s wear-leveling technique is in simple round-robin manner.

2.2 Transaction Process

Most of embedded applications including those of mobile phones either may not support multitasking or may not require concurrent access to the database. Especially, they may need multi-read operations, but they usually do not need multi-write operations. Thus, instead of supporting concurrency control, *LGeDBMS* supports only an atomic and durable update, which makes it possible to reduce unnecessary management cost.

To recover a system crash, *LGeDBMS* uses a PID mapping table versioning scheme for logging/recovery which is different from a traditional DBMS logging/recovery method. In the traditional logging method, when data is written to flash memory, a recovery log should be written to flash memory. It means that writing the log produces expensive additional I/Os. Moreover, the traditional recovery method causes other log reading I/Os and rolling back/forward I/Os(log-replay I/Os).

However, *LGeDBMS* logging method reduces the number of I/Os by writing a final PID mapping table once rather than writing a log for each data change. And recovery is performed simply by restoring the previous consistent PID mapping table, so log-replay I/Os are also reduced.

With the traditional DBMS, transaction begin/commit information should be logged. This scheme accompanies additional write operations, so that a DBMS application with only read operations receives unnecessary performance penalty. In case of read only transaction, we do not have to log about transaction information because we support only single transaction.

3. DEMONSTRATION

PIM component is a kind of middleware for personal information management, such as contact, schedule, and event, in mobile phones. As more people manage their personal information with mobile phones, mobile phones must guarantee data integrity and support various data formats. Therefore, the PIM component is implemented on *LGeDBMS* so that it can support transation/recovery and general schema. The PIM component has some additional features specialized for personal information. It provides common APIs and data structures which enable UI applications to access the data easily and safely. And it supports standard formats for personal data exchange such as vCard v2.1/3.0 and vCalendar v1.0[10]. The PIM component runs on our company's mobile phones and supports unified development environment for PIM application such as a phone book, a scheduler, and so on. Just as PIM component, specialized features for a specific application can be added on *LGeDBMS* as a form of plug-in. Figure 2 shows a PIM component architecture.

We demonstrate a PIM application (e.g. phone book) implemented on PIM component to show effectiveness of flash memory based *LGeDBMS*. The PIM application runs on: PC simulators to sniff its internal state, and a real phone to demonstrate real-time operation. Figure 3 shows the PC simulators. The former is a phone simulator which simulates functions of a mobile phone, and the later is a flash memory simulator which enables a disk to simulate a flash memory.

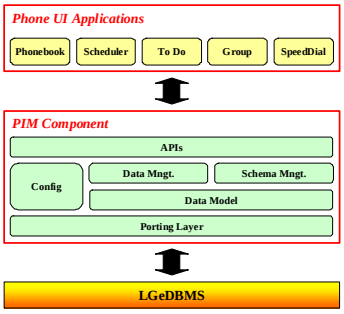

Figure 2: PIM Component Architecture

Figure 3: Simulators for Demonstration

4. REFERENCES

[1] Ban, A., Flash file system optimized for page-mode flash technologies, US patent 5,937,425, Filed Oct. 1997; Issued Aug. 1999; Assigned to M-Systems.

[2] Bobineau, C., Bouganim, L., Pucheral, P., and Valuriez, P, "PicoDBMS: Scaling down Database Techniques for the Smartcard," In *Proc. 26th Int'l Conf. on Very Large Databases*, pp. 11-20, Sept. 2000.

[3] Gal, E. and Toledo, S., "Algorithms and Data Structures for Flash Memories," *ACM Computing Surveys*, Vol. 37, No. 2, pp. 138-163, June 2005.

[4] IBM Corporation, DB2 Everywhere - Administration and Application Programming Guide, IBM Software Documentation, SC26-9675-00, 1999.

[5] Kawaguchi, A., Nishioka, S., and Motoda, H., "A Flash-Memory Based File System," In *Proc. Usenix Technical Conference*, Jan. 1995.

[6] Oracle Corporation, Oracle 8i Lite - Oracle Lite SQL Reference, Oracle Documentation, A73270-01, 1999.

[7] Rosenblum, M. and Pisterhout, J., "The Design and Implenmentation of a Log-Structured File System," In *Proc. 13th ACM Symposium on Operating Systems Principles*, Feb. 1992.

[8] Samsung Electronics, "Advantages of SLC NAND Flash Memory," http://www.samsungelectronics.com/

[9] Sybase Inc., Sybase Adaptive Server Anywhere Reference, Sybase Documentation, CT75KNA, 1999.

[10] vCard/vCalendar, "VCard and VCalendar," http://www.imc.org/pdi/