Xmas: An Extensible Main-Memory Storage System for High-Performance Applications

Jang Ho Park Yong Sik Kwon Ki Hong Kim
Sang Ho Lee Byoung Dae Park Sang Kyun Cha

School of Electrical Engineering, Seoul National University
Seoul, Korea, 151-742
{jhpark,yskwon,next,sangho,parkbd,chask}@kdb.snu.ac.kr

Abstract

Xmas is an extensible main-memory storage system for high-performance embedded database applications. Xmas not only provides the core functionality of DBMS, such as data persistence, crash recovery, and concurrency control, but also pursues an extensible architecture to meet the requirements from various application areas. One crucial aspect of such extensibility is that an application developer can compose application-specific, high-level operations with a basic set of operations provided by the system. Called composite actions in Xmas, these operations are processed by a customized Xmas server with minimum interaction with application processes, thus improving the overall performance. This paper first presents the architecture and functionality of Xmas, and then demonstrates a simulation of mobile communication service.

1 Introduction

Many applications, such as mobile communication service and process control, require high-performance access to database[7]. Disk-based DBMSs, however, are inadequate for this type of applications because of the latency of disk access for data retrieval and update. Main-memory DBMSs, which become feasible with the increasing availability of large and relatively cheap memory, are expected to support the above applications better than disk-based ones[4].

The main-memory DBMS architecture has been proposed and a few research prototype systems have been implemented[4]. Although these systems have been successful in proving the utility of the main-memory database system concept, most of them were not designed to be extensible enough to meet various application requirements.

Xmas is an eXtensible *Ma*in-memory *S*torage system, which has been developed at Seoul National University for high-performance database applications[10]. Xmas provides the core functionality of DBMS, such as data persistence, crash recovery, and concurrency control. Furthermore, it pursues an extensible architecture to deal with requirements from various applications.

The most important aspect of such extensibility is that an application developer can compose application-specific, high-level operations with a basic set of operations provided by the system. Called composite actions in Xmas, these operations are processed by a customized Xmas server with minimum interaction with application processes, thus improving the overall performance. The object-oriented design and implementation of Xmas facilitates this type of application-specific extension.

This paper is organized as follows. Section 2 presents the Xmas process architecture and functionality. Section 3 discusses the notion of composite action for application-specific extension. Section 4 describes the demonstration simulating mobile communication service. Section 5 concludes the paper.

2 The Xmas Architectural Overview

2.1 Process Architecture

As shown in Figure 1, the Xmas server process is multi-threaded to process several requests concurrently and to perform disk write for logging and checkpointing in parallel with normal transaction processing.

When the Xmas server starts up, it creates a pool of threads. A thread is not tied to any particular application process or transaction. When a request message arrives in the input queue, the action dispatcher thread converts it into an action object and enqueues it into the action queue. Action is the processing unit of Xmas, and from the perspective of the server process, a transaction is a sequence of actions from transaction begin to commit or abort. An action processing thread runs until it is blocked by waiting for lock or semaphore, or is preempted by thread scheduling. When it finishes executing an action object, it puts the result on the output queue and awaits another action.

The checkpoint thread and the log flush thread deals with disk I/O and run asynchronously with the other threads. The checkpoint thread moves dirty pages in the primary database to disk backup, and the log flush thread flushes the log tail in main memory to stable log volume.

2.2 Database Structure

The memory space of the primary database has a hierarchical structure. The primary database is divided into a number of fixed-size

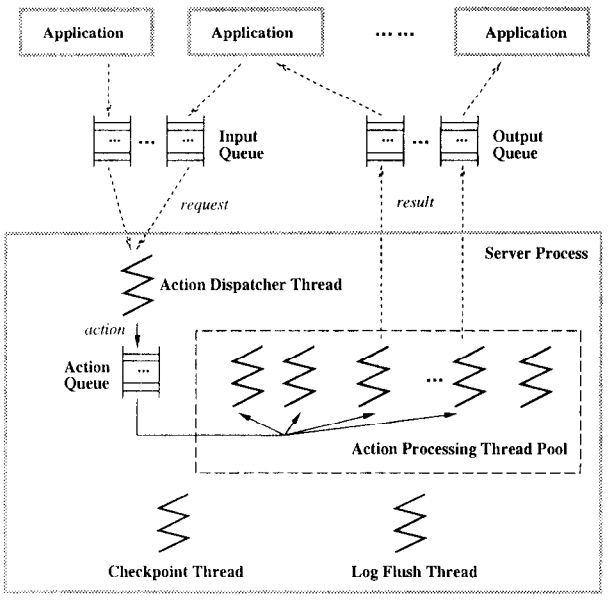

Figure 1: Xmas Multi-Threaded Server Process

segments, each of which is partitioned into pages. A page is divided into multiple slots. Segment is the unit of memory allocation and also the unit of checkpointing. Segments containing temporary objects are not checkpointed.

Figure 2 shows the objects of the primary database and their relationships. A container is a collection of records with same data structure. The pointer to a record is maintained through RID(Record IDentifier), which is composed of the segment number, the page number, and the slot number. A RID of a slot is unique and is converted into the virtual address of the slot by simple calculation. The catalogs maintain the control information of containers and indexes respectively.

Xmas supports hash index for exact value search, and T-tree[8] for range queries. An Xmas index stores only RIDs in its entries because the key value can be formed from the memory-resident record.

2.3 Recovery and Concurrency Control

Xmas supports ARIES[9] algorithm and fuzzy checkpointing[5]. The log consists of two parts: a memory-resident buffer and a stable log volume on disk. The memory-resident buffer holds the log tail, the most recently created part of the log. A battery-backup memory device[6] can be used as a stable log buffer between memory-resident buffer and disk to allow a transaction to commit without disk write. Performance measurement shows that the presence of the stable log buffer improves the throughput of update transactions by $2 \sim 10$ times[10]. Xmas also supports group commit for the environment without a battery-backup memory device.

Two-phase locking(2PL) protocol is used for the concurrency control of Xmas. The Priority inversion problem is resolved by priority inheritance protocol[1]. Xmas also limits the number of transactions executing concurrently to reduce the probability of aborting due to lock conflict. Xmas currently permits container-level locking. To support finer granularity, a table in relational database can be horizontally partitioned into a number of containers.

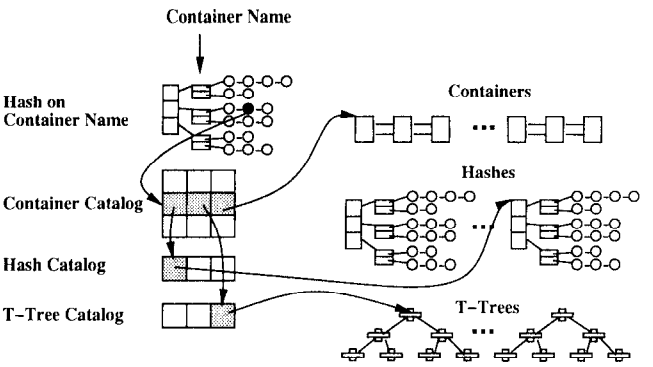

Figure 2: Logical Structure of a Database

3 Composite Action for Application-Specific Extension

Xmas is extensible in that the Xmas server can be customized to perform high-level operations of a specific application. Every operation processed by the Xmas server is modeled as an action, and an application-specific operation can be defined as a composite action with system-provided basic actions.

To facilitate the incorporation of composite actions, object- oriented approach is taken. Every action in Xmas is implemented as a subclass of Action class in which the common interface '*execute*' is defined. Each action has its distinctive arguments and implements the interface '*execute*'. Every action is processed through the same interface '*execute*' regardless of its content and hence facilitating the incorporation of composite actions.

Currently, we are designing an extension architecture based on composite action. The extension architecture consists of two major components: extension editor and action compiler. A system developer defines a composite action with the extension editor, which provides higher-level language like visual language for action composition. The action compiler preprocesses this composite action description, and generates the composite action definition and the API class definition. By compiling these definitions, Xmas is extended to understand application-specific operations.

4 Demonstration

To show the functionality and performance of Xmas, we implemented a demonstration system, which simulates mobile communication service(`http://kdb.snu.ac.kr/Xmas/mobile.html`). It is one of target applications of Xmas because high-performance access to subscriber information is required to serve call requests from subscribers. It has to keep track of each subscriber's location. For example, when a subscriber moves into a new registration area, his Home Location Register, i.e. the database maintaining his location information, updates his current location and forwards the subscriber profile to the Visitor Location Register of the new area.

Figure 3 shows the demonstration system architecture. There are two simulation programs running concurrently, each of which performs the same task of keeping track of each subscriber's location. Only the difference is the way of performing the task. One program uses system-provided basic actions, while the other uses newly-defined composite actions to show the performance benefit of composite action quantitatively.

A graphic user interface of Figure 4, implemented in Java, shows the current status and performance of simulation. This interface

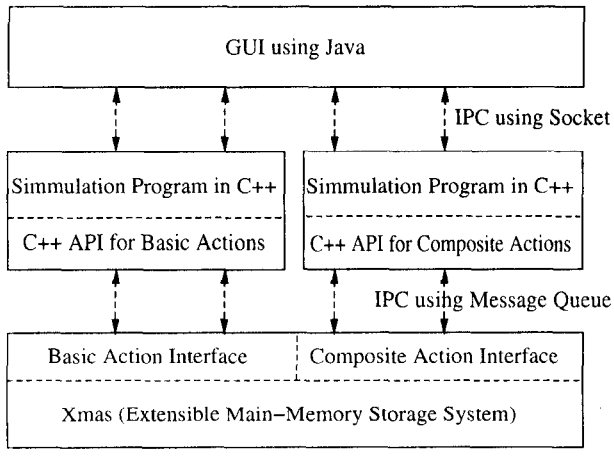

Figure 3: Demonstration System Architecture

provides a view of mobile communication environment. In this view, the movement, (de)activation, call, and hang-up of mobile phones is simulated graphically. Also shown is the statistics for database access including lookup, insert, update, and remove.

5 Status and Future Research

The Xmas prototype system is operational on Solaris 2.5. It consists of roughly 50,000 lines of C++ code with more than 150 classes. Currently, we are working on formalizing the notion of composite action and defining a language for composing composite actions. Employing Xmas as an embedded storage system for various applications, such as mobile communication service, process control[3], and geographic information systems[2] is also pursued.

Acknowledgment

This research has been partially supported by Engineering Research Center for Advanced Control and Instrumentation, and Pohang Iron and Steel Company in Korea.

References

[1] Robert Abbott and Hector Garcia-Molina. Scheduling Real-Time Transactions: A Performance Evaluation. In *Proceedings of the 14th VLDB Conference*, pages 1–12, 1988.

[2] Sang Kyun Cha, Ki Hong Kim, Chang Bin Song, Joo Kwan Kim, Joo Yong Jeon, and Yong Sik Kwon. A Middleware for Transparent Access to Multiple Spatial Object Databases. In *Proceedings of International Conference and Workshop on Interoperating Geographic Information Systems*, December 1997.

[3] Sang Kyun Cha and Jang Ho Park. An Object-Oriented Model for FMS Control. *Journal of Intelligent Manufacturing*, pages 387–391, October 1996.

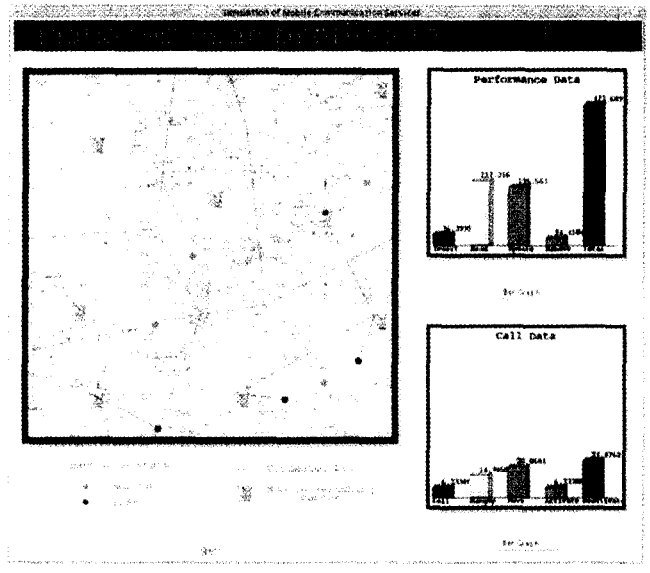

Figure 4: Demonstration System Interface

[4] Hector Garcia-Molina and Kenneth Salem. Main Memory Database Systems: An Overview. *IEEE Transactions on Knowledge and Data Engineering*, 4(6):509–516, December 1992.

[5] Robert B. Hagmann. A Crash Recovery Scheme for a Memory-Resident Database System. *IEEE Transactions on Computers*, C-35(9):839–843, September 1986.

[6] A. Haq. Non-Volatile Memory SIMM(NVSIMM), 1994. the specification for Axil's NVSIMM memory SIMM for Axil workstation 320 and S/400 product.

[7] Ben Kao and Hector Garcia-Molina. An Overview of Real-Time Database Systems. In S. H. Son, editor, *Advances in Real-Time Systems*, chapter 19, pages 463–486. Prentice Hall, 1995.

[8] Tobin J. Lehman and Michael J. Carey. A Study of Index Structure for Main Memory Database Management Systems. In *Proceedings of the 12th VLDB Conference*, pages 293–303, August 1986.

[9] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollback Using Write-Ahead Logging. *ACM Transactions on Database Systems*, 17(1):94–162, March 1992.

[10] Jang Ho Park, Byoung Dae Park, and Sang Kyun Cha. Xmas: An Extensible Main-Memory Storage System. In *Proceedings of 6th ACM International Conference on Information and Knowledge Management*, pages 356–362, November 1997.