Implementation Principles of File Management System for Omega Parallel DBMS

Mikhail L. Zymbler
Chelyabinsk State University
Russia
mzym@cgu.chel.su

Leonid B. Sokolinsky
Chelyabinsk State University
Russia
sokolinsky@acm.org

Abstract[1]

The paper describes development principles and the program structure of the Omega File Management System (OFMS) for the Omega parallel DBMS engine. The paper gives requirements for OFMS and the description of its general structure and components. The paper gives some effective protocol for interaction with the Disk Subsystem Unit and describes architecture of the Disk Subsystem Emulator. The paper also proposes a page replacement strategy based on so-called static and dynamic page ratings. Described OFMS was implemented for the MBC-100 massively parallel computing system.

1 Introduction

A parallel database management system (DBMS) design usually has some operating environment level in its structure. This level provides process management and scheduling, interprocess communication support, buffer management functions and the file management system.

The DBMS developer is faced with two approaches to implement such low-level services. The first, use these services from the operating system (OS) he or she works with, and the second, implement these services as part of DBMS. Current DBMSs usually use the latter alternative because general-purpose OS services usually are either too slow or inappropriate for database management needs [1].

This paper describes the Omega File Management System designed and implemented within the frames of the Omega Project [2]. The Omega Project is aimed to develop the parallel DBMS with hierarchical shared-nothing architecture for the MBC-100 multiprocessor computing system [3].

The Omega system hardware architecture corresponds to three-level hierarchy. The MBC processor module consisting of computation and communication processors presents first level. The second level corresponds to the Ω-cluster including four MBC processor modules connected each other and with Disk Subsystem Unit (DSU) by links. The third level of hardware hierarchy is presented by Ω-clusters composed into the Ω-system in shared-nothing manner. The detailed description of each level of the Omega system hardware architecture is given in [4].

The Omega Project Development Environment, its tools, cooperative development cycle are described in [5].

The Omega File Management System (OFMS) was created as a part of the Omega Operating System (OOS). The OOS includes also the following subsystems: the Thread Manager, the Ω-conductor and the Ω-router. The Thread Manager provides light-weighted processes (or threads) support. The Ω-conductor provides a message-passing system for communication within a cluster based. The Ω-router provides a message passing system for communication among clusters. Both the Ω-conductor and the Ω-router based on virtual channel mechanism. The interfaces and description of used algorithms of the above OOS subsystems are given in [4].

The rest of the paper is organized as follows. Section 2 describes the general structure of the Omega File Management System and requirements for it. Section 3 contains a description of the Disk Subsystem Emulator. In section 4 we give implementation principles of the Disk Manager and also and the protocol for data

[1] This work was supported by the Russian Foundation for Basic Research under Grant 00-07-90077.

Proceedings of the Workshop on Computer Science and Information Technologies CSIT'2000
Ufa, Russia, 2000

interchange with the Disk Subsystem Unit. The implementation principles of the Page Manager are given in section 5. A page replacement technique based on so-called static and dynamic page ratings is presented in section 6. Section 7 is devoted to development principles of the File Manager. Section 8 contains a conclusion.

2 General Structure of the Omega File Management System

The Omega File Management System provides connections between the Omega DBMS and the Disk Subsystem Unit. OFMS runs on each node of the cluster.

The main purpose of the OFMS is supporting concept of a file taken as a set of records of fixed length. Such files are used on the higher levels of system hierarchy to represent relations (tables), indexes and other components of a stored database.

The basic requirements for the OFMS are the following:

- OFMS should support files consisting of records of fixed length. Each file should have its ID unique within the given disk. Each record should have its ID unique within the given disk.
- OFMS should provide opportunity to introduce intra-file clustering on the higher levels of the system hierarchy. The Omega system does not support inter-file clustering.
- OFMS should support page buffering based on the uniform internal buffer pool. Access to file contents is possible by means of this buffer only.
- OFMS should support virtual shared-nothing architecture. It means that a separate virtual disk (disk pool) is assigned to each processor node of the Ω-cluster. One processor node can not address virtual disks (disk pools) of another.
- OFMS should not provide facilities to work with disks not belonging to the given Ω-cluster.
- OFMS should not support fragmentation and replication of files; fragmentation and replication are to be implemented on the higher levels of the system hierarchy. It means that each relation (table) fragment or replica corresponds to a separate and independent file.

The general structure of the OFMS is given in the Figure 1. OFMS includes the Disk Manager, the Page Manager and the File Manager.

The Disk Manager provides representation of a virtual disk as a set of numbered pages of 4 Kbytes length. In the future it is planned to add an opportunity to specify desirable page size for a file depending on available variants (for example, 4, 24 or 32 Kbytes).

The Page Manager provides representation of a stored database (more precisely its part) as a totality of page sets. Page set corresponds to a linked page list. The Page Manager allows to create and remove page sets, to add and remove page into/from a set and to access directly a page with a specified ID.

The File Manager provides database representation as a totality of files. Here file is a successive set of unstructured records of the same length (a record has only one *info* field). The File Manager allows to create and remove files, to add and remove records into/from a file, to edit record's contents (the value of info field), to access directly a record with a specified ID and also make iterators to cyclically return pointers to all records of a file without duplication.

3 Omega Disk Subsystem Emulator

The hardware platform of the Omega DBMS prototype development represents the MBC-100 multiprocessor system with four processor modules without a disk subsystem unit. Therefore there is a necessity to create the Omega *Disk Subsystem Emulator* (ODSE).

ODSE uses both electronic disks and disks of the Host-computer. At ODSE initialization there occurs an automatic reading of the database from the Host-computer disk and loading it into an electronic disk of some selected processor module. At a normal system shutdown there happens a reverse database dump from the electronic disk back to the Host-computer disk. During work some intermediate dumps to the Host-computer disk are possible.

For the Disk Subsystem Unit modeling a root processor module of the MBC-100 was allocated. This module is connected with Host-computer via a link. The Disk Subsystem Unit processor module is not used as a computational module. Thus the Ω-cluster was designed to use three processor modules and one Disk Subsystem Unit.

The ODSE encapsulates hardware features of the Disk Subsystem Unit. Such an approach allows to leave the same interface of the Disk Manager at porting to hardware configuration with real Disk Subsystem Unit, which, in turn, requires minimal modifications at porting the system as a whole.

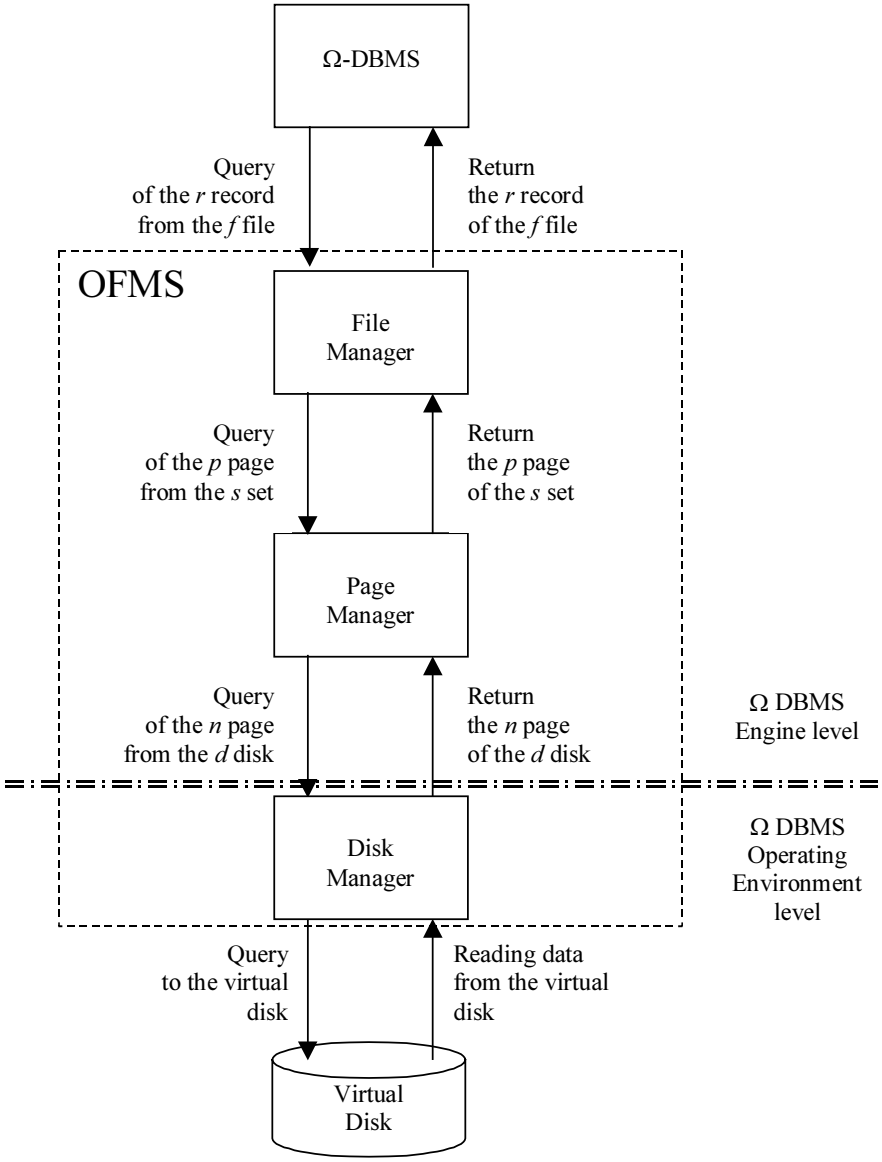

Figure 1. General structure of the Omega File Management System

4 Disk Manager

The Disk Manager provides basic input-output functions to organize data exchange with the Disk Subsystem Unit. The Disk Manager allows to abstract from hardware features of the Disk Subsystem Unit clusters and to consider it a virtual device consisting of some number of logical drives with page organization.

The number of disks equals the number of processor modules of the cluster. One disk is assigned to each processor module. The process can not access disks, which do not belong to it. Such approach simulates shared nothing architecture.

The page size is a parameter of the Omega system and equals to 4 Kbytes.

The Disk Manager implements virtual channel architecture. It means that via a link connecting the processor module with the Disk Subsystem Unit any number of read-write operations can be concurrently performed.

The Disk Manager was designed using client-server technique. The client's part of the Disk Manager runs on the every working node. And the server's part runs on the node of the disk subsystem unit.

Phase	Client

1.	if (link is locked by write) {
	stroke_counter--;
	break;
	}
	if (write operation via link is in progress) {
	stroke_counter--;
	break;
	}
	lock link by write;
	run writing header to the server;

2.	if (write operation via link is in progress) {
	stroke_counter--;
	break;
	}
	run writing info part to the server;

3.	if (header with code "operation done"
	has not come from the server) {
	stroke_counter--;
	break;
	}
	unlock link by write;
	stroke_counter=-1;

Phase	Server

1.	if (dump disk operation is in progress) {
	stroke_counter--;
	break;
	}
	if (client's link is locked by read) {
	stroke_counter--;
	break;
	}
	lock client's link by read;
	lock client's link by write;
	run reading header from the client;

2.	if (read operation via client's link is in progress) {
	stroke_counter--;
	break;
	}
	write page to the disk;
	run writing header with code "operation done" to the client;

3.	if (write operation via client's link is in progress) {
	stroke_counter--;
	break;
	}
	unlock client's link by read;
	unlock client's link by write;
	stroke_counter=-1;

Figure 2. Client and Server instruction phases of write page operation

The Server executes clients' queries to make some operation with the disk subsystem unit. Clients and the server exchange messages consisting of two parts – a header and an info part. The message header has the following fields: operation ID, operation type, page number and client's node number. Info part has no structure; it is a byte array length of which equals the page size. Message passing is carried out in two steps: message header first and then the info part.

Whenever a client starts any operation the descriptor of this operation is inserted into the operation table. Such a table is organized as a queue and its element has the following fields: operation ID, page number, client's node number, pointer to the buffer and operation state. The operation table is processed by the client's system thread. The factor-function of this system thread [4] activates the above thread only when asynchronous read-write operations via links are finished and the rest of the time this thread is disabled.

The server is made out as an independent process operating on a similar operation table in the disk subsystem node. This process cyclically carries out the following sequence of actions:

1) Wait for a message header from clients.
2) Process header, i.e. insert a new element into the operation table.
3) Process the operation table, i.e. execute possible actions (read page from disk, dump disk etc.).

Client and server use instruction phases ideology [4] to exchange data. Figure 2 shows client and server instruction phases of write page operation.

5 Page Manager

The Page Manager provides representation of a stored database as a totality of page sets. A page set corresponds to a linked page list. A page consists of a header and an *info* field.

- The Page Manager should satisfy to the following basic requirements:
- Page Set Allocation Table (SAT) should be supported. SAT is stored in a continuous disk area starting from the zero page.
- A special page set, the Free Space List (FSL) should be supported.
- An opportunity to add continuous blocks of pages to the set should be supported (for example up to 64 pages).
- Buffering of pages on the basis of uniform buffer space should be supported. The swap table (buffer pool index), containing numbers of all pages that are at the moment in the buffer should be supported. The page replacement algorithm should effectively work for completely filled buffer (it is a regular situation in case of database systems, - a page is ejected from the buffer when and only when it really is necessary for loading some other page).

- Access to contents of a page can be carried out only through its image in the memory.
- Page Identification (PID) should be supported. Such a PID should be unique within the disk. PID should provide direct access to a page. In an elementary case page number could be used as PID.
- The pre-fetch page selection should be supported.

Implementation of the Page Manager's main operations is based on the following subsystems: Buffer Manager, Buffer Directory, Free Space List Manager, Disk Directory Manager, SAT (Set Allocation Table) and Open Sets Manager.

The *Buffer Manager* provides allocation and releasing continuous memory chunks in the buffer pool using the Free Space List. The minimal unit of memory allocation is a 4 Kbytes block.

The *Buffer Directory* is used for organizing asynchronous operations of loading, saving and replacing page's images in the buffer pool. The Buffer Directory has the number of positions exceeding the number of pages can be placed into the buffer pool. This fact is essentially used for choosing replacement strategy (see 6).

The *Free Space List Manager* provides allocation of continuous page blocks and utilization of released pages.

The *Disk Directory Manager* provides functions to maintain the disk directory. The disk directory is situated in the zero page. It stores the free space list, the set allocation table and some other system information.

The SAT (*Set Allocation Table*) subsystem implements the set allocation table. Such a table is stored in the disk header and is sent to RAM simultaneously with initialization of the Page Manager. SAT's element contains the following information: the set identification, identifications of the set's first and last pages and some other system information.

The *Open Sets Manager* implements the table of open sets' descriptors.

6 Buffer Pool Management

Buffer pool management in the Omega DBMS supports:

- well-knowing page replacement strategies (e.g. LFU, FIFO, LIFO, LRU);
- dynamic selection of the best strategy for a specified page set and its access behavior;
- page privileges.

These principles are based on concepts of the redundant index of the buffer pool and page ratings.

The Buffer Directory mentioned above provides redundant index of the buffer pool. It means that size of Buffer Directory is $k*M$ where M is size of the buffer pool in pages and integer $k>>1$.

Redundancy of that index allows to store the history of loading pages into the buffer. Each Buffer Directory element has statistical attributes to provide well-knowing replacement strategies: reference counter, time of recent reference, time spending in buffer etc. These attributes change when system works. The Page Manager *dynamically* investigates the Buffer Directory to find replacement cycles. When a cycle is found the Page Manager selects the best replacement strategy for a specified page set and its access behavior.

Each element of the Buffer Directory (i.e. each page) has two attributes named static and dynamic ratings. The *static rating* is an integer from the [0;20] interval. User assigns the static rating at loading a page into the buffer. The static rating remains constant during a page is in the buffer and the static rating lost when the page is dropped from the buffer.

The *dynamic rating* is the function from the statistical attributes of the Buffer Directory, it's a real from the [0;1[interval. The Page Manager calculates the dynamic rating. The dynamic rating's value of a page may change during work.

The *summary rating* of a page is sum of the static and dynamic rating. If there is a necessity to make room for a new page then the page with minimal summary rating is to be dropped from the buffer. If there are several pages with minimal summary rating then their oldest page will be dropped.

Mechanism of the static rating realizes page privileges. We can divide all the pages into at least two classes: privileged that can remain in memory regardless of chosen replacement policy, and others. Mechanism of the dynamic rating allows to imitate any replacement strategy. For any two pages user can define an order of their replacement. This possibility is very important for a database recovery after crash.

7 File Manager

The File Manager provides database representation as a totality of files. Here a file is a successive set of records of the same length. In turn, a record consists of a header and an info field.

The basic functions of the File Manager are create/delete a file, open/close a file, append a record to a file, mark a record for deletion, remove all the records marked for deletion, fetch a specified record of a specified file.

Each file is placed in a separate page set. The FAT (*File Allocation Table*) stores information about correspondence between files and page sets. The FAT is placed in the continuous disk area starting from zero page.

8 Conclusion

This paper has presented implementation principles and the program structure of the Omega File Management System (OFMS) as part of the Omega Operating System (OOS). The OOS is used in parallel DBMS designed for the Russian MBC-100 massively parallel computing system. Requirements for the OFMS and description of its general structure and components were given.

Description of the Disk Subsystem Emulator and the effective protocol for interaction with the Disk Subsystem Unit were given.

The page replacement technique based on static and dynamic page ratings was proposed. Such a technique in fact allows implementation of any page replacement strategy.

The OFMS was implemented for the MBC-100 in C programming language. There are experiments with imitations various page replacement strategies now.

Described OFMS is mainly oriented on using in Omega project. However, it can be used in other similar applications with intensive disk interactions for the MBC-100.

References

1. Stonebraker, M. Operating System Support for Database Management. CACM 24:7 (July 1981) 412-418.
2. Sokolinsky, L., Axenov, O., Gutova, S. Omega: The Highly Parallel Database System Project. Proceedings of the 1st East-European Symposium on Advances in Database and Information Systems (ADBIS'97), St.-Petersburg. September 2-5, 1997, vol. 2. Nevsky Dialect (1997) 88-90
3. Zabrodin, A.V., Levin, V.K., Korneev, V.V.: The Massively Parallel Computer System MBC-100. Proceedings of PaCT-95. Lecture Notes in Computer Science, vol. 964 (1995) 342-356
4. Sokolinsky, L.B. Operating System Support for a Parallel DBMS with an Hierarchical Shared-Nothing Architecture. Proc. of the Third East-European Conf. on Advances in Databases and Information Systems (ADBIS'99), Maribor, Slovenia, September 13-16, 1999. Maribor University Publishing (1999) 38-45.
5. Zymbler, M.L. Computer Aided Design Facilities for Prototyping the Omega DBMS CSIT'99, Proceedings of the 1st International Workshop on Computer Science and Information Technologies, January 18-22, 1999, Moscow, Russia, vol. 2. MEPhI Publishing (1999)