Journal of Systems Architecture 68 (2016) 51–64

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

# A unified framework for designing high performance in-memory and hybrid memory file systems

Xianzhang Chen[a], Edwin H.-M. Sha[a,*], Qingfeng Zhuge[a], Weiwen Jiang[a], Junxi Chen[a], Jun Chen[a], Jun Xu[b]

[a] College of Computer Science, Chongqing University, Chongqing 400044, China
[b] Huawei Shannon Lab, Beijing, China

ARTICLE INFO

*Article history:*
Received 30 November 2015
Revised 31 March 2016
Accepted 22 May 2016
Available online 24 May 2016

*Keywords:*
In-memory file systems
Hybrid memory file systems
Non-volatile memory
Flash memory
Performance

ABSTRACT

The emerging non-volatile memory technologies provide a new choice for storing persistent data in memory. Therefore, file system structure needs re-studying and re-designing. Our goal is to design a framework that gives high-performance in-memory file accesses and allows a file whose data can be stored across memory and block device. This paper presents a novel unified framework for in-memory and hybrid memory file systems based on the concept that each file has a contiguous "File Virtual Address Space". Within this framework, the file access for in-memory data can be efficiently handled by address translation hardware and the virtual address space of file. The file accesses for data in block devices are handled by a dedicated page fault handler for file system. A file system called Hybrid Memory File System (HMFS) is implemented based on this framework. Experimental results show that the throughput of HMFS approaches the memory bus bandwidth in best cases. Compared with in-memory file systems, HMFS reaches 5 times, 2.1 times, and 1.6 times faster than EXT4 on Ramdisk, RAMFS, and PMFS, respectively. Compared with EXT4 on SSD and EXT4 using page cache, HMFS also achieves 100 times and tens of times performance improvement, respectively.

© 2016 Elsevier B.V. All rights reserved.

## 1. Introduction

The growing real-time requirements for data processing push the development of in-memory computing. Currently, many platforms and applications directly store data in memory to close the huge gap between memory and traditional storage, such as HANA [1] and Spark [2]. Existing DRAM-based computing systems need to back up in-memory data on persistent storages, such as magnetic disks and flash memory. The backup procedures may cause large overhead and degrade the system performance.

Recently, emerging Non-Volatile Memory (NVM) [3–5] techniques provide opportunities for reserving data in high speed memory persistently. Thus, no backups are required. NVMs [6–8] show advantages as the promising candidate of persistent storage. For example, domain wall memory [8] and PCM [9] show high density, near-DRAM speed, byte-addressable, shock resistance, and low power leakage. The experiments show that the file access for in-memory data can be 100 times faster than the file access for the data stored in flash memory. Therefore, an in-memory file system can be designed to take advantages of NVM for achieving high performance file accesses.

The in-memory file systems, however, have a size problem especially for large files. The NVM may be too expensive to reserve all the data in a system. It is also not necessary to reserve all the data of a large file in NVM for many applications. Therefore, we need to use hybrid storages in a system, including NVM, flash memory, and magnetic disk, to efficiently maintain persistent data. To the authors' knowledge, none of the existing in-memory file systems can allow the primary data of a single file are stored across memory and other devices.

As the proposed file system enables a file to be stored across memory and block devices, the size problem of memory can be alleviated. Based on the design, the cold part of a file can be stored in block devices while the hot part is stored in the memory. Compared with the existing in-memory file systems, we can share the memory for more files in the proposed file system. The applications can easily benefit from the high performance of the memory and the large capacity of block devices simultaneously.

In this paper, our goal is two-fold. First, we try to design a framework to provide highly efficient file accesses for in-memory file systems. The framework is designed to fully exploit the

* Corresponding author.
*E-mail addresses:* xzchen.cq@gmail.com (X. Chen), edwinsha@gmail.com (E.H.-M. Sha), qfzhuge@gmail.com (Q. Zhuge), jiang.wwen@gmail.com (W. Jiang), cjxsimon@gmail.com (J. Chen), newdays.chenjun@gmail.com (J. Chen), xujun09@huawei.com (J. Xu).

http://dx.doi.org/10.1016/j.sysarc.2016.05.004
1383-7621/© 2016 Elsevier B.V. All rights reserved.

benefits of NVM. Second, we try to extend the framework from memory to block devices such that the primary data of a single file can be stored across hybrid storages. Within the framework, a hybrid memory file system is expected to benefit from both the high speed of NVM and the large capacity of block devices.

Considering the file accesses for in-memory file data, the challenge is how to efficiently locate the physical locations of data pages using memory management hardware. We propose a novel framework of "file virtual address space" for highly efficient file accesses over the file data in the memory. In this framework, each file has its own continuous virtual address space. For a file in the framework, the data pages are organized by a hierarchical page table dedicated to the file, which is called "file page table". When a file is opened, its file virtual address space is embedded into the kernel virtual address space. Different from the traditional file systems, data accesses to any location of the opened file in the framework are performed efficiently by the hardware Memory Management Unit (MMU) rather than searching the metadata structure by software. The virtual address space of a file is established by inserting a few pointers of the file page table into the corresponding entries of the kernel page table.

Considering the file accesses for file data stored on hybrid storages, the challenging problem is how to efficiently decide if the file data is in the memory or the block device. The proposed framework of "file virtual address space" can be extended and applied to solve the problem. In the unified framework, each file also has its own continuous virtual address space. For a single file in the unified framework, it is possible to store part of its data on the main memory and the rest on the block device. The data pages on the main memory and the data blocks on the block device of the file, however, are organized by a same file page table dedicated to the file. For in-memory file data, the file accesses take advantages of the hardware MMU in CPU with high performance. For the file data stored on the block devices, a dedicated page fault handler is designed to efficiently complete the file accesses.

A new file system named Hybrid Memory File System (HMFS) is designed and implemented based on the new unified framework. HMFS is implemented in Linux to support general file system interfaces, parallel file accesses of multiple threads, and various executables. We evaluate the performance of HMFS with standard benchmarks. Experimental results show that the performance of both sequential and random reads/writes of HMFS are significantly improved compared with a set of existing file systems, including EXT4 on Ramdisk [10], RAMFS [11], and the state-of-the-art in-memory file system PMFS [12]. The throughput of HMFS approaches the memory bus bandwidth in the best cases. To the authors' knowledge, this is the best known result for file systems in the literature.

The main contributions of this paper include:

- We propose a new unified framework of file virtual address space for designing high-performance in-memory file systems and hybrid memory file systems for memory and block devices. Base on this framework, each file has its own continuous virtual address space. A file system can take advantages of hardware MMU to achieve highly efficient file accesses.
- We design and implement a high-performance hybrid memory file system, HMFS, based on the proposed framework. A hierarchical file page table is designed to organize file data on memory or block devices. The proposed HMFS uses MMU hardware and a dedicated page fault handler for fast address mapping and file access.
- Experiments are conducted to compare the throughput of HMFS with other file systems, including EXT4 on Ramdisk, RAMFS, and PMFS. Experimental results show that HMFS outperforms these file systems. The throughput of HMFS even approaches the memory bus bandwidth in the best cases. Compared with EXT4 on SSD and EXT4 using page cache, HMFS also shows significant performance improvement.

The rest of the paper is organized as follows. The related works are described in Section 2. We present the design framework and implementation of HMFS in Section 3. We then evaluate the performance of these file systems in Section 4. Finally, Section 5 concludes the paper.

## 2. Related work

NVM has been widely used in different levels of storage system designs. In the device level, due to the higher read/write speed and the better wear property than flash memory, NVM are used as disk cache to achieve higher performance and extend the lifetime of flash memories [13,14]. Such an improved device can be integrated into the hybrid file system.

In the level of system software, NVM is generally used to store file data or metadata. The idea of using memory as storage has long been investigated, such as RAMFS [11], TMPFS [15], and Phoenix [16]. A typical type of file system using NVM is in-memory file systems. In-memory file systems use NVM as the sole permanent storage for the whole file system [12,17–21]. A direct approach is to deploy existing block-based file systems on NVM, such as the mobile computing systems with Linux-like operating systems. For these file systems, the optimizations for the space utilization of NVM has been studied in [19,20,22]. BPFS [17] uses a tree to organize the file data. BPFS developed atomic and ordered update techniques for NVM. MRAMFS [18] is mainly designed for small files. PMFS [12] is the stat-of-the-art in-memory file system. PMFS uses a B-tree to organize the data pages of a file.

These in-memory file systems cannot fully utilize the hardware MMU to achieve high performance. They need to search metadata structures for the physical locations of file data by software. SCMFS [21] utilizes contiguous virtual address space for file accesses. However, SCMFS has large overhead for the management of the mapping table and the virtual address space of the files. In this work, we present a framework of file virtual address space that can fully take advantages of MMU hardware with nearly zero overhead.

Another type of file system using NVM is hybrid memory file system. On one hand, many previous works use NVM as a buffer for file data or metadata on block devices, such as magnetic disk and Solid State Disks (SSDs). In [23–25], the NVM is used to cache the metadata or hot data written to block devices. The dirty data stored in NVM-based cache are flushed back to block devices only when the cache is full. On the contrary, the dirty data stored in volatile cache should be periodically backed up. Thus, the NVM-based cache can improve the performance of the system. On the other hand, many hybrid memory file systems are proposed to use NVM as a storage for file data or metadata. In [26–31], all the metadata of a file system are reserved in the NVM. The file data are stored on block devices. The performance of file accesses is improved as the time for search metadata structures is reduced. However, these file systems do not fully utilize the fast speed of NVM and the hardware MMU to further improve the performance of data accesses. TridentFS [32] proposes a framework to place file data and metadata on hybrid storages, such as NVM, SSD, and magnetic disk. The metadata structure of TridentFS is a multi-level index and the file accesses should also search through the index by software. Existing in-memory or hybrid memory file systems either stores the whole file in the memory or in the block device. None of these file systems allow a file whose data can be stored across memory and block device.

Download English Version:

https://daneshyari.com/en/article/457610

Download Persian Version:

https://daneshyari.com/article/457610

Daneshyari.com