An Adaptive Directory Accelerating Mechanism for NVM-based File Systems*

Extended Abstract†

Ben Trovato‡
Institute for Clarity in Documentation
Dublin, Ohio
trovato@corporation.com

G.K.M. Tobin§
Institute for Clarity in Documentation
Dublin, Ohio
webmaster@marysville-ohio.com

Lars Thørväld¶
The Thørväld Group
Hekla, Iceland
larst@affiliation.org

Valerie Béranger
Inria Paris-Rocquencourt
Rocquencourt, France

Aparna Patel
Rajiv Gandhi University
Doimukh, Arunachal Pradesh, India

Huifen Chan
Tsinghua University
Haidian Qu, Beijing Shi, China

Charles Palmer
Palmer Research Laboratories
San Antonio, Texas
cpalmer@prl.com

John Smith
The Thørväld Group
jsmith@affiliation.org

Julius P. Kumquat
The Kumquat Consortium
jpkumquat@consortium.net

ABSTRACT

This paper provides a sample of a LaTeX document which conforms, somewhat loosely, to the formatting guidelines for ACM SIG Proceedings.[1]

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

KEYWORDS

ACM proceedings, LaTeX, text tagging

ACM Reference Format:
Ben Trovato, G.K.M. Tobin, Lars Thørväld, Valerie Béranger, Aparna Patel, Huifen Chan, Charles Palmer, John Smith, and Julius P. Kumquat. 1997. An Adaptive Directory Accelerating Mechanism for NVM-based File Systems: Extended Abstract. In *Proceedings of ACM Woodstock conference (WOODSTOCK'97),* Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.475/123_4

*Produces the permission block, and copyright information
†The full version of the author's guide is available as `acmart.pdf` document
‡Dr. Trovato insisted his name be first.
§The secretary disavows any knowledge of this author's actions.
¶This author is the one who did all the really hard work.
[1]This is an abstract footnote

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
WOODSTOCK'97, July 1997, El Paso, Texas USA
© 2016 Copyright held by the owner/author(s).
ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

1 INTRODUCTION

Emerging non-volatile memory (NVM) technologies such as spin-torque transfer, phase change, resistive memories and 3D XPoint technology promise offering both byte-addressable I/O performance and persistent memory storage, which revolutionize the architecture design of file systems. As a result, various novel file systems based on NVM have been proposed, such as PMFS[], SCMFS[], NOVA[] and HMFS[]. NOVA[] and HMFS[] are designed by combining faster volatile DRAM and slightly slower, but persistent non-volatile memory, and promise to offer efficient data access.

The directory is a vital part of accelerating data access in file systems. Because they have a large impact on application performance[nova].The design of directory balances the trade-off between directory write and read. One design involves multi-level namespaces, another design is based on full name index. Multi-level namespace directory(MND) sacrifices read efficiency because of recursive name lookups, but supports fast write such as RENAMEs and CPs. On the contrary, full name directory(FND) offers fast lookups without recursive scan, but brings high write overhead if a part of full name changes. For traditional disk file system with block I/Os, the small directory name write causes heavy write overhead because of write amplifications[] and large random write. Therefore, traditional disk file systems such as ext4 and ext3, use MND as its directory design. Besides, betrFs[] provides FND design use LSM tree, which is able to combine small random write into large sequential write. But betrFs still has slow file deletion and renames[]. Thus, MND is the best choice for disk file system design.

Non-volatile memory brings opportunities to solve FND heavy write overhead problem. NVM offers short write latency, high efficient random access, and byte-addressability. Disk write amplification is not exist in NVM. However, NVM based file systems are still designed following traditional MND architecture, which is not suitable for byte-addressable NVM. Besides, although nvm solves the problem of write amplification in FND, strictly full naming directory still causes large amount of random write if directory name

changes. Too much write increases overhead and has a bad impact on system performance.

To address these problems, we propose ADAM, adaptive directory accelerating mechanism for hybrid nvm-based file systems, which offers fast read as well as small write overhead. ADAM contains two parts: adaptive full directory namespace (AFDN) and self-adaptive namespace strategy(SANS). The key idea of ADAM is that: 1) we firstly record access frequency for both read and write for each directory; 2) we also record the directory size state according to the number of subfiles; 3) we divide directory into different levels based on directory access frequency and size, and then use state-map to mark each directory level; 4) we allocate AFND areas to store name index entries for both directories and files. The index entries is random stored in an AFND area. Each AFND area has root directory, root directories record strictly full pathname, and other directories in each area record path name compared with specified area root directory. Root directory is important for file system performance. Because when root directory name changes, it would not cause changes of other directory path names. We pick up root directories according to directory levels marked in state-maps. In AFDN, directories which is write hot, read hot or has a large number of subfiles are most possible to be root directories. The level of directories might be changed as system is running. Therefore, we change AFND areas to keep file system maintaining the most efficient data access performance. We call it self-adaptive namespace movement strategy. It involves three movements of AFDN areas: split, merge and inherit. For example: a write hot directory A might split out of its original AFND areas. And when A becomes cold and small, it would be merge into its parent AFND area. Thus, this self-adaptive namespace strategy(SANS) ensures that the namespace root directory is selected for reasonableness and minimize the write overhead of AFND while the file system is running.

We implement our AFND and SANS in DAFS based on NOVA[], a hybrid DRAM/NVMM file system. We improve index structure in Nova and build a high efficient hybrid index which is more suitable for hybrid DRAM/NVMM storage. NOVA originally build radix trees for indexing all directories in both DRAM and NVM, which takes up quit amount of expensive DRAM. The central idea of directory indexing in DAFS is to make full use of large capacity NVM and access faster DRAM. We build a hash table for each AFND area in NVM and a radix tree index for all AFND root directories. We evaluate the performance using micro- and macro-benchmarks.

The contributions of this paper are:

• We identify the old, unsuitable directory design (MND) in byte-addressable NVM based file systems. MND sacrifices read efficiency in order to avoid large random small writes, which is more suitable for traditional disk file systems. We analysis the write overhead of strictly full name directory(FND): small name writes and large random write latency. Small write causes write-amplification problem in disk file systems. Large random write latency would slow down the performance in both disk and nvm based file systems. We propose AFND, an adaptive full namespace directory design for NVM based file systems, which offers fast lookup as well as reduces write overhead compared with strictly FND.

• We proposed a novel strategy SANS, a self-adaptive namespace strategy to keep the AFND file system maintaining the most efficient directory access while the file system is running. SANS ensures the reasonableness of AFND root directory and minimize the write overhead in case of the states of directories are changed.

• We implement a efficient hybrid DRAM/NVM directory index: RADIX tree in DRAM and hash tables in NVM. This hybrid index utilizing the persistence and large-capacity NVM and access faster DRAM.

• We implement ADAM, an adaptive directory accelerating mechanism, in NOVA, a hybrid DRAM/NVM file system[], which might be the fastest nvm-based file system. the evaluation results show that XXXX.

The remainder of this paper is organized as follows. Section II provides background. Section III and Section IV presents the design and implementation of AFND and SANS. We aslo show the hybrid index of directory in DRAM/NVM based nova. We discuss the evaluation results in Section VI. Finally, we provide related work in section VII and conclude our paper in Section VIII.

Table 1: Comparison of different memory characteristics[]

Category	Read latency	Write latency	Write Endurance	Random accessing
DRAM	60ns	60ns	10^{16}	High
PCM	50~70ns	150~1000ns	10^{9}	High
ReRAM	25ns	500ns	10^{12}	High
NAND Flash	35μs	350μs	10^{5}	Low

2 BACKGROUND AND MOTIVATION

2.1 Non-Volatile Memory

Computer memory has been evolving rapidly in recent years. A new category of memory, NVM, has attracted more and more attention in both academia and industry[]. NVM provides features like byte-addressability, non-volatility,and higher-to-flash speed. Table 1 shows the performance characteristics of different memory technologies. NVM provides slightly longer read latency compared with DRAM, while its write latency is apparently longer than DRAM. Similar to NAND Flash, the write endurance of NVM is limited, especially for PCM. Thus, many hybrid DRAM/NVMM file systems have been proposed to support lower latency, high write endurance, and persistence. Balancing writes and reads in hybrid DRAM/NVMM system is critical for software system design. Besides, NVM has high performance for random accessing like DRAM, which is different from traditional Flash.

2.2 Previous Directory Structures

Previous Block-I/O based file systems such as F2FS, ext4 use multi-level directory design, in order to reduce block I/O write overhead compared with FND. However, this design sacrifices read speed for involving recursive scans of multi directory-namespaces. Strictly full directory namespace brings a large amount of write overhead because of write-amplification in block-I/O based file systems, which do not exist in hybrid DRAM/NVMM based file systems. Btrfs 0.2[] introduced zones to BetrFS, a disk-based file system. The main idea of BetrFS 0.2 is first dividing directories and files into different zones according to the size of each directory and file, and then use LSM tree to reduce I/O overhead brought by renames. BetrFS 0.2 reduces

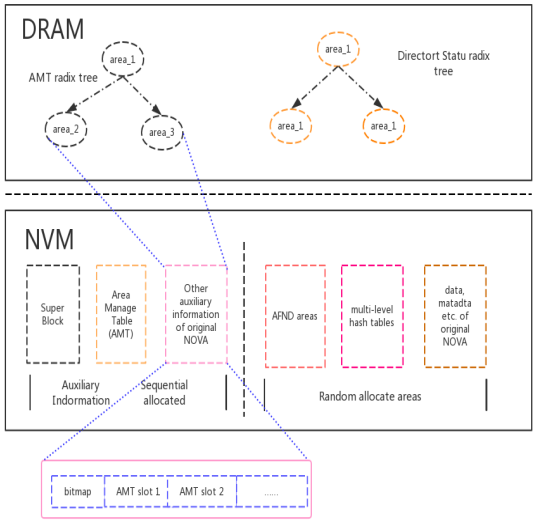

Figure 1: Architecture of ADAM-NOVA

a part of potential rename overhead by only in the consideration of size. However, it is not the most import reason that affecting read and write during system runtime. Because access frequency directly affects the performance. For example, read/write frequency directory is more qualified to be a new zone than seldom access but large directory B.

Although NVM provides byte-addressable I/O that solve the write amplification of block-based I/O. Newly NVM-based file systems such as pmfs, nova, and hmfs still use the traditional directory design of disk-based file system, which is apparently not suitable for NVM storage. Directly introducing strictly full name directory into NVM-based file systems is not appropriate. It causes a large number random writes and corresponding write overhead in low write endurant NVM. Another challenge is that files and directory show different characteristic in access frequency during runtime which can be used to balance read and write overhead to enhance the system performance. But there are no consideration of access adaptive state in existing nvm-based file systems like nova and pmfs, which use the same standard for each directory and file.

3 DESIGN

Our design of ADAM in NVM-based file system aims to enhance directory read efficiency and minimize its write overhead. In this section, we first present the overview design of adaptive directory accelerating mechanism. We then describe AFND, our layout for directory namespace. Finally, we describe SANS, self-adaptive namespace strategy, which keeps file system maintaining high data access efficiency, even though the state of directories changes during file system running.

3.1 ADAM Overview

To accelerate directory access and suit to byte-addressable non-volatile memory, we introduce ADAM to replace the old directory design in NVM-based file systems. We design ADAM based on three observations: First, existing NVM-based file systems use traditional directory design MND, which is suitable for block-I/O based file systems, but not the byte-addressable NVM-based file system. Second, while nvm solves write-amplification problem in FND, too much write still increase system latency. Third, during system runtime, directories show different characters in read-frequency, write-frequency, and size, which should be considered into file system optimization.

Based on these observations, we conceptually develop ADAM from two aspects: storage and movementïijŇwhich are showed in figure 1. We introduce AFND area, an adaptive full namespace directory areas to manage the storage of directories. Each AFND area not only stores the entries of both directories and files but also records the state of directories with a novel design-state_map. Based on the state recorded in AFND, we develop SANS, a self-adaptive namespace strategy to manage movements of AFND areas. SANS ensures the consistent performance during system lifetime.

3.2 AFND Area

AFND area contains two parts: directory and file entry storage area and state_map area. We balance the directory area trade-off[] and initialize each AFND area as a 512KB block array. Each directory and file entry is initially aligned on a 128-Byte boundary. The AFND area root directory records area_ID. Each AFND area_ID is marked by the hash value of its root directory strictly full name and indexed by hybrid index. In AFND storage part, the locality benefits are less important in NVMM-based storage, because both NVMM and DRAM support high qualified random access. ADAM append new directory entry to each zone in a round_robin order, so that directory entries are evenly distributed among AFND areas. During file system runtime, the statement of directories might be changed. ADAM identifies different statements of directories based on read frequency, write frequency and directory size. We define three levels of statement for directories hot, warm and cold. Different access frequency and size correspond to different states which are described in table 1.

State_map is a two-bits bitmap, which records statements of entry slots in AFDN area which is shown in table 2. Compared with other data-intensive data structures[], State_map not only saves storage space but also incorporates the role of a bitmap for marking valid/invalid states of slots. We make a trade-off with read frequency, write frequency and size to calculate directory state. Read frequency is an important statement component. If the whole AFND area is read hot, then it would save the lookup time and accelerate the read efficiency. Because write overhead is the primary reason that affects system performance. Thus, we consider write frequency as the most important factor in calculating statements of directories. Because the pathnames of all files and directories stored in the same AFND area are the adaptive full names compared with its root directory. In strictly FND design, if one directory name changes, then all the path name that contains this directory must be overwrite. However, in AFND, if the name of a root directory changes, the only root directories which contain the name would be affected. The rest directories and files in AFND remain the same, which will minimize the write overhead. Size is also one of the

factors in calculating the statement of directories. Because larger directories are most likely to generate a large amount of write overhead, once name change happens. We show the rules of calculating directory statement in table 1. The real-time access frequency and the number of subfiles are recorded in dram radix tree in order to fully utilizing the short latency of DRAM. The state_map in NVM updates from time to time.

AFND defines the layout of both directory- and file-inode stored in NVMM. In order to update status for each directory, each AFND owns a novel two-bits bitmap called state_map ahead as figure 3 shows. State_map records entry valid information and status information of every slot, which is the base of self-adaptive namespace strategy.

3.3 SANS Model

We divide files and directories into three types: NORMAL_FILE, NORMAL_DIRECOTRY, and ROOT_DIRECTORY. In our mechanism, the AFND root directory has two strengths compared with normal directories: 1)it is fastest to find the subfiles of root directories. 2) The name changes of root directories have the least effect on the write overhead of the entire system. Therefore, the division of the area has a great influence on the system performance. ADAM introduce SANS, a self-adaptive namespace strategy to manage AFND areas selection. To enhance read performance and reduce write overhead, directories which are read hot, write hot and large should become the AFND area. These root directories which are cold and small should merge back to AFNF areas as NORMAL_DIRECTORY. We also define the boundaries of directories and AFND areas, shown in table 3.

During system lifetime, characters of directories might change. We choose the number of subfiles and subdirectories, read-frequency and write-frequency as the most important characters. Because of the changes happened during system lifetime, we find it is necessary to move AFND areas to maintain a well performance consistently . Because 1) new large directories with frequent access might appear which is suitable to be root driectories in order to enhance system performance; 2) the slots in existing AFND areas takes up large NVMM space but the root directories are seldom visited by users; 3) subdirectory is stronger than its parent root directory, which means the vast majority visits to the AFND root directory are to visit this strong subdirectory. Due to these reasons, SANS defines three movements for various AFND areas: split, merge and inherit. The system executes these movements according to the status recorded in state_map, which is shown in table 2.

Split has two state: positive split and negative split. Splitting is the way to create new AFND areas and select corresponding root directories throughout system lifetime. When split happens, all the subfiles and subdirectories are copied to the new AFND area. The Root directory is still staying in the original AFND area with recording its AFND area_ID. Positive splitting happens when the system periodically checks the state_map and finds that directories with high access frequencies are waiting to be split. AFND areas root directories chosen by positive splitting are either with high read frequencies and large size or with high write-frequencies and large size. Therefore, positive split improves the efficiency of finding files or directories and reduces the overhead of writing. The MAX size of each AFND area is fixed and initialized to 512KB[]. When a directory becomes unusually large or there are not enough free slots in AFND, we introduce negative split. When negative splitting happens, we tend to select large size directories as new root directories. By splitting large size directories, we not only free slots in the AFND area, but also reduced the amount of potential write overhead.

Merge happens when AFND areas become too small and the access frequency is too low. If the number of valid slots in this AFND area is less than 1/5 of the entire region, then we assume that it satisfies the merging size. We check the root directory from its state_map to determine that whether the AFND area satisfies the low access frequency request of merging. AFND areas that exhibit features including small size and low access frequency take up a significant amount of nvmm space and they can neither accelerate the read access of directories and files nor contribute to the reduction of write overhead. Usually, the size of directories is reduced when deletion happens. Therefore, we check AFND area after large deletions happen and merge areas if it meets the necessary conditions of merging. During merging, we copy all the file- and directory-entries back to their parent AFND area. And then free this AFND area and make it reusable.

Inherit define an AFND movement that a hot and large subdirectory replace its parent root directory, which is small and cold. We call this kind of subdirectory Strong Subdirectory. During our experimenting, we find there is a big challenge against splitting. We assume AFND area A a root directory P, and it has a strong subdirectory S. In ADAM we check the state_map in A and then find S is the write-frequency and large directory that satisfies splitting requirement. Thus, we split S as a new AFND area B. After moving all subfiles and subdirectories belong to S, the parent root directory P becomes small directory and seldom visiting one, which meets the requirement of merging. We then merge AFND area A and free it. The whole steps are Check A-> Find S-> Allocate B-> Splitting S-> Merging A-> Free B. The overhead of managing NVM space is very high under these circumstances. To solve this problem, we introduce inherit. When we find a strong subdirectory of a root directory, we copy the rest of subfiles and subdirectories back to its parent AFND area and let this strong subdirectory replace the original root directory. The steps of inherit are Check A->Find S-> Copy only a small part of entries. Inherit reduces the number of allocations and releases in NVM at a finer granularity, and improves the system performance.

4 IMPLEMENTATION

We implement ADAM in NOVA, a log-structured file system based on Hybrid DRAM/NVMM. Note that ADAM can also be implemented in other NVM-based file systems such as PMFS[] or HMFS[] to accelerate directory accessing. This section describes the implementation details of ADAM on NOVA.

4.1 Hybrid Index

We pay close attention to directory index for hybrid DRAM/NVMM storage. In hybrid DRAM/NVMM file systems, DRAM has lower latency but low capacity, whereas NVM has large capacity but a

slightly larger access latency. Therefore, the index in hybrid memories file system should utilize the strength of both lower latency DRAM and larger capacity NVM. However, it is not well balanced in NOVA. NOVA build a radix tree in DRAM for each directory inode[], which takes too much space of DRAM. This index structure is obviously unrealistic in practical applications when there are many directories. Encouraged by HiKV, we implement a more efficient hybrid directory index structure in NOVA, which is shown in figure 3.

AFND hash table We build a multi-level hash table for each AFND area. Multi-level hash tables are able to solve the hash conflicts, which is well-test in F2FS[]. Unlike the multi-level hash tables in F2FS, five-level hash tables are enough for an AFND area in our experiment, because each AFND has a fixed number of slots for directories and files. Each hash entry contains a hash value of every directory inode and the position of AFND slot as a hash pair designed as figure 4. To reduce the hash table size, each hash entry contains a valid bit and we reuse invalid hash entries for new files and directories. Hash tablea greatly improve the speed of name searching of directories and files.

Area manager table(AMT). We allocated AMT in the index area of NOVA in NVMM. AMT is initialized as a 4KB block and is divided into 54-Byte slots. We use an AMT bitmap to manage slots allocating and releasing. AMT manages informations of all the AFND areas and related hash tables. Each AMT slot corresponds to a AFND area and records its area_ID as well as hash table position. We mark each AMT slot by area_ID as the searching key. Therefore, users can easily find specific AFND area and hash table through AMT.

DRAM index. AMT is most commonly accessed in searching directory inodes stored in AFND areas, which is suitable for high read efficiency DRAM. We keep a radix tree in DRAM and the leaves of the radix tree store information as AMT slots stores. We use a radix tree because there is a mature, well-tested, widely-used implementation in the Linux kernel[]. Throughout DRAM radix tree searching, we can easily and quickly find the right AFND. The way to keep consistence between radix tree and AMT is same as Nova does. Besides, we also implement a status radix tree for each AFND area and is pointed by AMT radix tree leaves. The status radix tree records the access frequency state and size state for each accessed directory. We periodically convert the state stored in status radix tree to the value stored in state_map. We use a status tree because the state of directories is often changed and read during system run time. And the state information stored in radix tree would not take too much space of DRAM.

With our hybrid indexing, users can easily and quickly find the directory and file inode by the following step: 1) first calculate the AFND area_ID using path name; 2) find the correct AMT slot; 3)find the position of AFND area and corresponding hash table; 4)find directory inode position by the hash key-value pair. Compared with the original index design in Nova, our hybrid index greatly saves DRAM space, and also enhance the lookup efficiency through a hash table.

4.2 Scalable Name Storage

In our mechanism, the space overhead is a challenge for full name storage. Strictly full naming involves large duplicate path name storage which caused large space overhead. Our adaptive full naming has dramatically reduced the space overhead, compared with strictly full naming. However, challenges still remain in our mechanism. First, adaptive full names of all files and directories still contribute to a large space overhead. Second, each AFND slot has the boundary of 128 Byte which is enough for common directories and files. However, if the depth of directory is very deep or the path name is very large, the AFND slot is not enough to store their adaptive full name.

In order to solve these challenges, we implement scalable name storage. In our mechanism, the adaptive full name of directories is frequently used, such as calculating AFND area_ID when splitting, merging and inheriting movements happen. Thus, all directories store their adaptive full name in the slot. However, for normal_files it is not necessary to store their adaptive full name. Therefore, we store their own name and the parent directory position in the 128-byte slot. We save a large NVM space and reduce corresponding write overhead by this way. When the name is too large to store under the boundary of 128 Bytes, we will extend this slot. We scan for a free slot in this AFND area to store the large name, and make a pointer pointed to this extended slot in the original directory or file slot. If the extended is still not enough, then we extend another slot and make a pointer, and so on.

By implementing this scalable naming method, we not only reduce the space overhead, but also satisfied the large name storage requirement.

4.3 Consistence And Garbage Collection

In our mechanism, only the small but frequent access data, such as AMT entries and state of directories, is stored in DRAM, in order to enhance performance of file system. This part of data is updated in place at runtime with a journal log implemented in nova. The journal log will record the movement and data in a 4KB buffer. Once movements of AFND area are started. NOVA prefer FAST GC with its original directory design, which is not conflicting with our mechanism.

5 EVALUATION

6 RELATED WORK

BetrFs 0.2 Since both MND and FND have strength and shortage on balancing read and write overhead. Researchers have been conducted on how to utilize the high-speed-read of FND and low-overhead-write of MND. BetrFS 0.2[] builds zones for directories and files using write-optimized dictionaries[] and improves the file system performance based on betrFS[]. They propose zones inspired from Dynamic subtree partitioning and implement it on BetrFS. The division of zones mainly refers to the size of files and directories. The files or directories which are chosen as roots of zones would potentially reduce the risk of large write overhead.

BetrFS 0.2 proposes a mechanism to maintain a consistent rename and scan performance trade-off in the consideration of possible

size changes during system lifetime. The mechanism includes split and merge, and they define the Max size (ZoneMax) and Min size (ZoneMin) to decide when to split and when to merge. However, there are several problems remaining in BetrFS 0.2. 1) BetrFS 0.2 is designed for disk-based storage, which means the shortage of block I/Os remains in BestrFS. Small number of writes would cause large write overhead because of the write amplification problem. BetrFS 0.2 uses write-optimized dictionaries to combine the small random writes into large sequential writes which not suitable for NVM with high qualified random access. 2) The detection of zones is only based on the size of both directories and files, which is not accurate in zone-root selection. Size is only the potential reason that affects system read and write overhead. In the contrast, access frequency directly reflects read/write preference of the system. There are no considerations of access frequency in BetrFS 0.2. 3) The split and merge in BetrFS 0.2 might cause redundant NVM allocate, release and write. Therefore, in our mechanism, we introduced inherit to prevent this situation. 4) The space overhead from storing names is large. Our ADAM optimized this problem in implementing ADAM-NOVA.

HiKV. Hybrid DRAM/NVMM based file systems provide a good architecture to fully use the strength of faster but volatile DRAM and non-volatile but slightly slower NVMM. It is reasonable to design an efficient index for hybrid DRAM/NVMM file system. In order to reduce the system latency, original nova overuse the capacity of DRAM via putting radix trees for each file and directory. It enhances system performance but takes up to much DRAM space. HiKV[] introduces a reasonable and efficient hybrid index for hybrid DRAM/NVMM KV-store systems. Hikv builds hash tables of kv pairs in NVMM and build a B-tree in DRAM to quickly search for hash tables. We improve this hybrid index and implement it in the nvm-based file system. However, B-tree is mainly used to reduce I/Os in disk based storage system which is not suitable for main memories. Our mechanism introduces radix trees in dram which is more suitable for Linux kernel and uses multi-level hash tables to avoid hash conflicting. Therefore, our ADAM-nova optimizes space management and remains high-qualified indexing.

7 CONCLUSION

This paper proposes ADAM, an adaptive directory accelerate mechanism designed for NVM-based file systems to enhance the directory read/write. ADAM includes AFND, an adaptive full namespace directory design and SANS, a self-adaptive namespace strategy. AFND introduces a novel state_map to record changing access-status and offers a reasonable management of both directory-inodes and file-inodes. SANS makes file system keep a efficient and persistent performance during runtime. We also implement a hybrid index with a good balance of DRAM and NVMM. Out measurements show XXXXXX.

REFERENCES

[1] Rafal Ablamowicz and Bertfried Fauser. 2007. CLIFFORD: a Maple 11 Package for Clifford Algebra Computations, version 11. (2007). Retrieved February 28, 2008 from http://math.tntech.edu/rafal/cliff11/index.html

[2] Patricia S. Abril and Robert Plant. 2007. The patent holder's dilemma: Buy, sell, or troll? *Commun. ACM* 50, 1 (Jan. 2007), 36–44. https://doi.org/10.1145/1188913.1188915

[3] American Mathematical Society 2015. *Using the amsthm Package.* American Mathematical Society. http://www.ctan.org/pkg/amsthm.

[4] Sten Andler. 1979. Predicate Path expressions. In *Proceedings of the 6th. ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages (POPL '79).* ACM Press, New York, NY, 226–236. https://doi.org/10.1145/567752.567774

[5] David A. Anisi. 2003. *Optimal Motion Control of a Ground Vehicle.* Master's thesis. Royal Institute of Technology (KTH), Stockholm, Sweden.

[6] Mic Bowman, Saumya K. Debray, and Larry L. Peterson. 1993. Reasoning About Naming Systems. *ACM Trans. Program. Lang. Syst.* 15, 5 (November 1993), 795–825. https://doi.org/10.1145/161468.161471

[7] Johannes Braams. 1991. Babel, a Multilingual Style-Option System for Use with LaTeX's Standard Document Styles. *TUGboat* 12, 2 (June 1991), 291–301.

[8] Malcolm Clark. 1991. Post Congress Tristesse. In *TeX90 Conference Proceedings.* TeX Users Group, 84–89.

[9] Kenneth L. Clarkson. 1985. *Algorithms for Closest-Point Problems (Computational Geometry).* Ph.D. Dissertation. Stanford University, Palo Alto, CA. UMI Order Number: AAT 8506171.

[10] Jacques Cohen (Ed.). 1996. Special issue: Digital Libraries. *Commun. ACM* 39, 11 (Nov. 1996).

[11] Sarah Cohen, Werner Nutt, and Yehoshua Sagic. 2007. Deciding equivalances among conjunctive aggregate queries. *J. ACM* 54, 2, Article 5 (April 2007), 50 pages. https://doi.org/10.1145/1219092.1219093

[12] Bruce P. Douglass, David Harel, and Mark B. Trakhtenbrot. 1998. Statecarts in use: structured analysis and object-orientation. In *Lectures on Embedded Systems*, Grzegorz Rozenberg and Frits W. Vaandrager (Eds.). Lecture Notes in Computer Science, Vol. 1494. Springer-Verlag, London, 368–394. https://doi.org/10.1007/3-540-65193-4_29

[13] Ian Editor (Ed.). 2007. *The title of book one* (1st. ed.). The name of the series one, Vol. 9. University of Chicago Press, Chicago. https://doi.org/10.1007/3-540-09237-4

[14] Ian Editor (Ed.). 2008. *The title of book two* (2nd. ed.). University of Chicago Press, Chicago, Chapter 100. https://doi.org/10.1007/3-540-09237-4

[15] Simon Fear. 2005. *Publication quality tables in LaTeX.* http://www.ctan.org/pkg/booktabs.

[16] Matthew Van Gundy, Davide Balzarotti, and Giovanni Vigna. 2007. Catch me, if you can: Evading network signatures with web-based polymorphic worms. In *Proceedings of the first USENIX workshop on Offensive Technologies (WOOT '07).* USENIX Association, Berkley, CA, Article 7, 9 pages.

[17] David Harel. 1978. *LOGICS of Programs: AXIOMATICS and DESCRIPTIVE POWER.* MIT Research Lab Technical Report TR-200. Massachusetts Institute of Technology, Cambridge, MA.

[18] David Harel. 1979. *First-Order Dynamic Logic.* Lecture Notes in Computer Science, Vol. 68. Springer-Verlag, New York, NY. https://doi.org/10.1007/3-540-09237-4

[19] Maurice Herlihy. 1993. A Methodology for Implementing Highly Concurrent Data Objects. *ACM Trans. Program. Lang. Syst.* 15, 5 (November 1993), 745–770. https://doi.org/10.1145/161468.161469

[20] Lars Hörmander. 1985. *The analysis of linear partial differential operators. III.* Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], Vol. 275. Springer-Verlag, Berlin, Germany. viii+525 pages. Pseudodifferential operators.

[21] Lars Hörmander. 1985. *The analysis of linear partial differential operators. IV.* Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], Vol. 275. Springer-Verlag, Berlin, Germany. vii+352 pages. Fourier integral operators.

[22] IEEE 2004. IEEE TCSC Executive Committee. In *Proceedings of the IEEE International Conference on Web Services (ICWS '04).* IEEE Computer Society, Washington, DC, USA, 21–22. https://doi.org/10.1109/ICWS.2004.64

[23] Markus Kirschmer and John Voight. 2010. Algorithmic Enumeration of Ideal Classes for Quaternion Orders. *SIAM J. Comput.* 39, 5 (Jan. 2010), 1714–1747. https://doi.org/10.1137/080734467

[24] Donald E. Knuth. 1997. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms (3rd. ed.).* Addison Wesley Longman Publishing Co., Inc.

[25] David Kosiur. 2001. *Understanding Policy-Based Networking* (2nd. ed.). Wiley, New York, NY.

[26] Leslie Lamport. 1986. *LaTeX: A Document Preparation System.* Addison-Wesley, Reading, MA.

[27] Newton Lee. 2005. Interview with Bill Kinder: January 13, 2005. Video. *Comput. Entertain.* 3, 1, Article 4 (Jan.-March 2005). https://doi.org/10.1145/1057270.1057278

[28] Dave Novak. 2003. Solder man. Video. In *ACM SIGGRAPH 2003 Video Review on Animation theater Program: Part I - Vol. 145 (July 27–27, 2003).* ACM Press, New York, NY, 4. https://doi.org/99.9999/woot07-S422

[29] Barack Obama. 2008. A more perfect union. Video. (5 March 2008). Retrieved March 21, 2008 from http://video.google.com/videoplay?docid=6528042696351994555

[30] Poker-Edge.Com. 2006. Stats and Analysis. (March 2006). Retrieved June 7, 2006 from http://www.poker-edge.com/stats.php

[31] Bernard Rous. 2008. The Enabling of Digital Libraries. *Digital Libraries* 12, 3, Article 5 (July 2008). To appear.
[32] Mehdi Saeedi, Morteza Saheb Zamani, and Mehdi Sedighi. 2010. A library-based synthesis methodology for reversible logic. *Microelectron. J.* 41, 4 (April 2010), 185–194.
[33] Mehdi Saeedi, Morteza Saheb Zamani, Mehdi Sedighi, and Zahra Sasanian. 2010. Synthesis of Reversible Circuit Using Cycle-Based Approach. *J. Emerg. Technol. Comput. Syst.* 6, 4 (Dec. 2010).
[34] S.L. Salas and Einar Hille. 1978. *Calculus: One and Several Variable.* John Wiley and Sons, New York.
[35] Joseph Scientist. 2009. The fountain of youth. (Aug. 2009). Patent No. 12345, Filed July 1st., 2008, Issued Aug. 9th., 2009.
[36] Stan W. Smith. 2010. An experiment in bibliographic mark-up: Parsing metadata for XML export. In *Proceedings of the 3rd. annual workshop on Librarians and Computers (LAC '10)*, Reginald N. Smythe and Alexander Noble (Eds.), Vol. 3. Paparazzi Press, Milan Italy, 422–431. https://doi.org/99.9999/woot07-S422
[37] Asad Z. Spector. 1990. Achieving application requirements. In *Distributed Systems* (2nd. ed.), Sape Mullender (Ed.). ACM Press, New York, NY, 19–33. https://doi.org/10.1145/90417.90738
[38] Harry Thornburg. 2001. Introduction to Bayesian Statistics. (March 2001). Retrieved March 2, 2005 from http://ccrma.stanford.edu/~jos/bayes/bayes.html
[39] TUG 2017. Institutional members of the TEX Users Group. (2017). Retrieved May 27, 2017 from http://wwtug.org/instmem.html
[40] Boris Veytsman. [n. d.]. acmart—Class for typesetting publications of ACM. ([n. d.]). Retrieved May 27, 2017 from http://www.ctan.org/pkg/acmart