The Name of the Title Is Hope

Ben Trovato*
G.K.M. Tobin*
trovato@corporation.com
webmaster@marysville-ohio.com
Institute for Clarity in Documentation
Dublin, Ohio, USA

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, USA
cpalmer@prl.com

John Smith
The Thørväld Group
Hekla, Iceland
jsmith@affiliation.org

Julius P. Kumquat
The Kumquat Consortium
New York, USA
jpkumquat@consortium.net

ABSTRACT

A clear and well-documented LaTeX document is presented as an article formatted for publication by ACM in a conference proceedings or journal publication. Based on the "acmart" document class, this article presents and explains many of the common variations, as well as many of the formatting elements an author may use in the preparation of the documentation of their work.

CCS CONCEPTS

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

KEYWORDS

datasets, neural networks, gaze detection, 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. 2018. The Name of the Title Is Hope. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

As DRAM performance has reached its limits, several new nonvolatile memory (NVM) concepts have been proposed in recent years and significant progress has been made. The existing computer system structure puts forward higher requirements for memory systems, such as ultra-low power consumption computing, high density and low cost data storage. Emerging NVMS with high performance, good scalability, and new capabilities may become important technical support factors to meet these requirements. In this context, the emerging class of non-volatile memory (NVM) promises to revolutionize the memory hierarchy.

Recently, many applications have been improved to run on nonvolatile memory. In this context, the improvement of general software for non-volatile memory is particularly important, after all, all software is based on general software. Hence the allocator for nonvolatile memory. The dynamic allocation of persistent memory is used to build high-performance applications, from index structures, to transactional memory, to in-memory database systems. Memory allocators are generally well tuned for volatile memory (for example, DRAM) to achieve low latency, high scalability, and low fragmentation. Allocators designed for persistent memory need to maintain the distinctive characteristics of DRAM allocators for high-performance memory management. More importantly, they should make better use of the features of non-volatile memory so that they run well when running on non-volatile memory.

In addition to the above advantages, non-volatile storage also has the disadvantage of inconsistent read/write speed and limited write times. Since nonvolatile memory has two sides, the advantages should be used to cover the disadvantages when designing its general software. Many allocators [] are designed for persistent memory, and they need to manage the persistent heap through various types of metadata and their allocation algorithms to represent the unique design of non-volatile memory and efficiently serve memory allocation and release. In addition to the normal allocator requirements, non-volatile allocators usually need to meet the requirements of crash consistency, high performance, and reliability.

The "`acmart`"

*Both authors contributed equally to this research.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

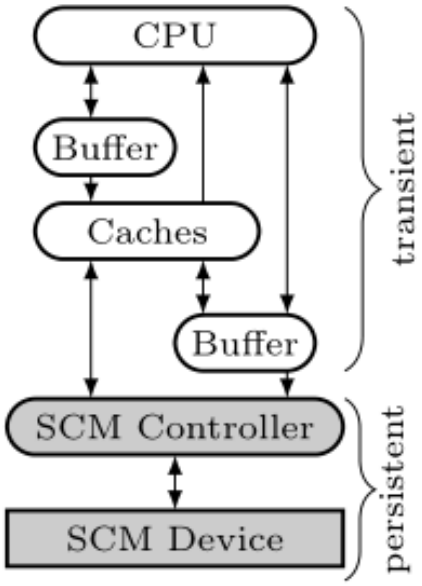

Figure 1: Storage chain in X86

2 BACKGROUND

The universal memory allocator has long been a fundamental building block of software, and the need for dynamic memory management is also essential in non-volatile memory. In order to use NVM effectively, it must integrate effectively with the existing system architecture, requiring changes at the operating system and application level. However, due to the change of operating medium, this brings many differences and challenges to the design of general purpose software.

2.1 Terminology

Data consistency. NVM is managed using an NVM-enabled file system that grants the application layer direct access to the NVM through memory mapping, which enables the CPU to directly use load and store statements for data access. The path from the NVM to the CPU registers is long and mostly volatile. As shown in Figure 1, it includes storage buffers, CPU caches, and memory controller buffers, all of which the software has little control over. In addition, modern cpus implement complex out-of-order execution and partial storage sort (Intel x86) or loose memory sort (IBM PowerPC). Therefore, memory stores need to be explicitly sorted and persisted to ensure consistency. Current x86 cpus provide CLFLUSH, MFENCE, SFENCE, CLFLUSHOPT, CLWB, and non-temporary storage instructions to enforce memory sorting and data persistence. CLFLUSH synchronization ejects cached lines and writes them back to memory. Therefore, the data needs to be aligned with the cache line boundaries to avoid accidental ejection of incorrectly shared data. SFENCE is a memory barrier that serializes all pending storage, while MFENCE serializes pending load and storage. Non-temporary storage bypasses the cache by writing to a special buffer that will be ejected when the buffer is full or when SFENCE is issued. CLFLUSHOPT is an asynchronous version of CLFLUSH that requires serializing the MFENCE. Finally, CLWB writes the cache row back to memory without evicting it, which is useful when the data is accessed shortly after persistence.

Data security and recovery. Many existing NVM system software uses a transaction logging mechanism to keep fault atoms updated. When an application uses an allocator, the allocation and release requests need to be handled in a special way, and the allocation and release of the object itself must be delayed until the object is confirmed to be error-free in execution. At the same time, the allocator needs to handle partial writes during power outages and memory leaks under different conditions. This includes the need to secure data throughout the allocator's operation. When the program restarts, it loses the previous address space, thus invalidating any stored virtual Pointers. Therefore, you need to design ways to discover and recover the data stored in the NVM.

2.2 Differences and Challenges

Memory is a key component of computer systems, and their capabilities and integration with systems have evolved greatly over time. In fact, a whole set of devices and technologies have been developed. The integration of nonvolatile memory with traditional storage architectures is also an issue worth considering, from an architectural perspective, the insertion of a given NVM technology anywhere in the memory hierarchy based solely on its characteristics in terms of technical process, bandwidth, and durability. From a volume perspective, integration will depend on a number of cost considerations. In general, the higher the performance and cost of a given memory technology, the closer the integrated CPU.

At present, there are two main approaches to NVM integration, one is as external memory, which becomes the upper level of SSD storage and the next level of DRAM storage. This integration takes advantage of the NVM access speed and provides caching for the next level of SSDS. Second, for upper-level DRAM, there is no need to modify too much system software. Because NVM is very similar to SSD, they are both non-volatile and only need to make some minor changes to the existing software based on the access characteristics of NVM. Another way of integration is to use memory as a storage medium with the same structure as DRAM. This integration mode can make more effective use of the performance advantage of NVM, the data access form of NVM is the same as DRAM, the CPU can directly access it through the data bus. This article discusses the differences and challenges of allocator design in the latter integrated approach.

First of all, the most notable feature of NVM and DRAM is its non-volatility, and for this feature, the allocator needs to meet the crash consistency. Failure to update the allocator metadata may result in inconsistent metadata. We classify this inconsistency as an inconsistency within the allocator. Such internal inconsistencies often have disastrous consequences, such as failure to reboot properly, misallocating memory objects currently in use, or leaking large chunks of memory. Failure inconsistencies may also occur on the allocator, which are external inconsistencies. When an application allocates memory through malloc, a failure occurs when the address is returned to the application after the data within the allocator has been persisted. This causes the allocator to think it

has allocated memory but not use it, essentially causing a memory leak.

Second, because the NVM preserves the data storage nature in any case, the allocator's memory efficiency becomes even more important. For NVM, each memory leak or memory fragment will not disappear with a shutdown or unexpected failure, but will remain in the memory used by the allocator. To make the allocator work well on non-volatile memory, we need to design its allocation mechanism and defragmentation algorithm more carefully, reducing the overhead of memory leaks and memory fragmentation. In addition, the design needs to be modified for memory inefficiencies in the old allocator, such as memory operations that want to be shut down to eliminate the impact, which the existing design needs to do away with and provide a more efficient solution.

However, for NVM, the difference in write times compared to DRAM also leads to a large divergence in its allocation strategy. Since the number of writes available to NVM is small, we must make the number of writes to memory as even as possible. Within the allocator, due to the nature of metadata access, data within the stack and heap is accessed multiple times, resulting in an extreme imbalance in write volume. In addition to the writing of internal metadata, the writing of common data also presents a big difference due to the cold and hot data. If allowed to write, the lifetime of the NVM will be greatly reduced. Therefore, the equalization of write volume is also a factor to be considered in the allocator design.

Finally, there is the issue of recoverability of the allocator data. In the case of NVM, the allocator is guaranteed to recover quickly because the data is not lost due to a power outage. This involves how to design the allocator's internal data structures and recreate their previous state after a crash by establishing a recovery mechanism.

3 NVM ALLOCATOR DESIGN

The design of the allocator can be divided into three types: 1) Focus on crash consistency and data consistency optimization(Makalu[], nvm_malloc[]). 2) Focus on optimizing performance by leveraging NVM features(NValloc[], PAlloc[]). 3) Focus on the write imbalance caused by the optimization of NVM write times(UWLalloc[], WAFA[]). Next, we'll look at the design principles of the NVM allocator in order of three.

3.1 Makalu and Nvm_malloc

Makalu, a system that addresses non-volatile memory management. Makalu provides an integrated allocator and recovery time garbage collector that maintains internal consistency, avoids NVRAM memory leaks, and is highly efficient in the event of a failure. By relying on offline garbage collection during failover, the persistence overhead per allocation is greatly reduced. A typical small-object persistence allocation does not require any data to be flushed to persistent memory because all relevant metadata can be effectively reconstructed from the object graph during recovery.

Makalu uses an integrated allocation release mechanism and garbage collection, based on the Boehm Demers Weiser garbage Collector (bdwgc). Makalu and bdwgc differ in several key ways. In addition to the transitions needed for the allocator to become fail-safe and run correctly during a restart, there are some major

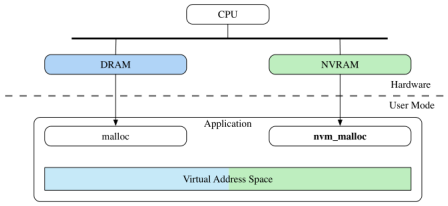

Figure 2: Operating mode

differences between Makalu and bdwgc in their allocation and release strategies. Because bdwgc is designed for online automatic memory management, it has poor support for explicit release. Since we only support GC offline, explicit online release is very important in Makalu. Although the bdwgc supports thread-local allocation, it does not support thread-local release. Makalu has chosen to use persistent metadata. To reduce the cost of persistent metadata updates, Makalu maintains a list of free objects in temporary memory during the online phase. Allocating and de-allocating simply remove and add the corresponding memory objects from the temporary free list, respectively. Failure may cause a temporary loss of unfinished memory objects in the Makalu short free list. Makalu uses offline garbage collection to reclaim these objects and fixes all other external inconsistencies caused by failures based on the accessibility of memory objects in the persistent heap. As a positive side effect, it also reclaims persistent memory that has leaked due to programming errors. Makalu is given its public interface. To achieve transaction-based integration, Makalu provides an interface for delaying release requests until the person in use can confirm that the released object is indeed inaccessible.

nvm_malloc, which is a generic memory allocator concept used in the NVRAM era as a basic building block for persistent applications. It introduces the concept of managed named allocations to simplify recovery, and uses a combination of volatile and non-volatile memory to provide high performance and fail-atom allocations. nvm_malloc is a fast, flexible, and secure general-purpose memory allocator on NVRAM that takes advantage of the expected coexistence of volatile and non-volatile memory in the unified virtual address space to achieve atomicity and performance of operations. The allocator is the central point of recovery and integrates file system-like naming concepts for allocations, thereby reducing the burden on applications to implement their own concepts. nvm_malloc is used in parallel with the volatile allocators, both of which run on the same virtual address space, as shown in Figure 2. nvm_malloc links other data structures through the root object, and the application just needs to make sure that the root object is accessible through ids and the rest of the objects are accessible through Pointers. The overall design of VMM-Malloc is based on the popular jemalloc allocator, which we chose for its excellent scalability and overall performance. A series of virtual addresses are "reserved" during the initialization phase of nvm malloc using mmap and anonymous mapping modes. The various guard flags in the mmap call ensure that the reserved area can neither be written nor read from. By default, nvm malloc reserves a virtual address range of 10TiB for this purpose. mmap can now be safely used to map files on NVRAM to reserved scope.

3.2 NValloc and PAlloc

NVAlloc is designed to improve memory consumption due to repeated cache line flusher and small random access in persistent memory and static board isolation. First, NV Alloc eliminates cache row refresh by mapping contiguous blocks of data in the board to interleaving metadata entries stored in different cache rows. Second, it writes small metadata units to the persistent account log in sequential mode to remove random heap metadata access from persistent memory. Third, it supports slab deformation rather than using static slab separation, which allows slabs to be converted between size grades, resulting in significantly higher slab usage.

For smaller allocations (<16 KB), NV Alloc implements arena and tcache to reduce thread contention. Each CPU core has an arena and each thread has a tcache. Each thread will be assigned to an arena with the fewest number of allocated threads. When the user frees a small block, the worker thread will first use the R tree to find its size class. It then returns to its corresponding tcache. NV Alloc uses an interleaving map of board bitmaps and an interleaving layout of tcache to avoid cache line flushing when small heap metadata access is needed. For large allocations, NVAlloc records the large allocations with WAL, puts the metadata into DRAM and backs it up in NVM for speed. WAL's layout is staggered to avoid cache row flushes.

PAllocator, a fail-safe non-volatile allocator designed to emphasize high concurrency and capacity scalability. Contrary to previous work, PAllocater completely addresses the important challenge of persistent memory fragmentation by implementing an efficient defragmentation algorithm. It consists of two allocators, one for small block allocation and the other for large block allocation, where metadata is persisted in a hybrid SCM-DRAM tree. In addition, the allocator emphasizes the importance of persistent memory fragmentation and presents an efficient defragmentation algorithm. PAllocator uses multiple files instead of a single pool, making it easy to expand and shrink a persistent memory pool. PAllocator uses three different allocation strategies for small, large, and large allocation sizes, most of which are independent of each other to ensure robust performance for all allocation sizes. To defragment memory, it takes advantage of the punch function of sparse files. To provide quick recovery, PAllocator retains most of its metadata and relies on a hybrid SCM-DRAM tree to trade off performance and recovery time when necessary.

3.3 UWLalloc and WAFA

NVM has the potential to be a new generation of memory, however, the limited write durability presents serious challenges. A number of wear equalization techniques have been proposed to alleviate this problem from different angles. In fact, there is a serious wear equalization problem in the allocator because of the large write times difference. Some allocators are optimized for this.

UWLalloc is a wear equalization aware memory allocator that (1) always tends to allocate less written memory blocks on memory requests, and (2) temporarily does not allow allocation of blocks that exceed a threshold. In addition, the allocator provides a unified management scheme for stack and heap areas for the first time, enabling better balancing of writes in the stack and heap areas. Traditional dynamic memory allocators, such as the glibc allocator,

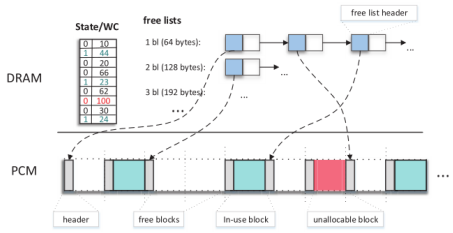

Figure 3: structure of UWLalloc

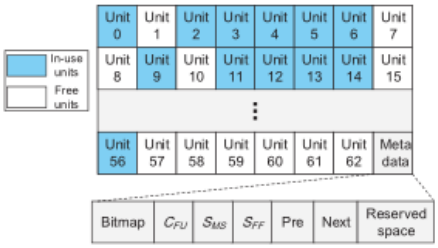

Figure 4: Illustration of WAFA

pair frequently changing metadata with data to facilitate the allocation and release of memory space management, and cache newly freed small blocks of memory for priority allocation. As a result, severe memory write offsets occur when using these allocators. As shown in Figure 3, UWLalloc takes the isolated free-list approach, which maintains a set of linked lists, each of which contains free blocks of a specific size. The memory space allocated in UWLalloc is 64-byte aligned, so each memory block allocated consists of a base memory block (64 bytes), and each free list is a multiple of 64 bytes in size. A traditional allocator stores the list pointer in the header and footer of the freed block. These in-place Pointers are updated when adjacent free blocks are merged or large free blocks are split, resulting in a large number of small writes. To prevent NVM from making these frequent small writes, UWLalloc demots the list pointer to DRAM and links all the list nodes to their associated NVM memory region.

WAFA is a wear equalization aware Fine grain distributor (WAFA) for NVM. WAFA divides pages into basic memory units to support fine-grained updates. WAFA rotates the page's base memory unit to distribute fine-grained updates evenly across memory cells. The fragmentation of the base memory units per page resulting from the memory allocation and release operations is reorganized through the reorganization operation. The basic structure is shown in Figure 4. In order to provide an efficient attrition allocator for fine-grained memory requests, the main idea of WAFA is to divide the page into smaller sections and alternate the units of new requests. Based on this idea, WAFA proposed the design of page metadata, optimal clock allocation strategy. Basically, WAFA divides a memory page into cells with a minimum allocation size. For smaller subpages, the minimum allocation size is the same as the size of the last level of cache row. This causes write magnification by interfering with the bits of other subpages. In this article, the size of the unit is set

to 64 bytes. WAFA proposes a clockwise best fit (CBF) strategy to allocate cells for fine-grained memory requests. The main idea of CBF is twofold. First, we allocate the most suitable continuous free cell for the memory allocation request, through which the memory cell has the highest chance of being written after allocation. Second, the units are assigned in clockwise order.

4 SUMMARY

Emerging NVMS offer a wide range of performance, maturity, and scaling potential. Emerging NVM offers to improve or replace existing storage, simplify the storage hierarchy, design new architectures, and enable new capabilities. However, there are still some obstacles to the practical use of NVM. This is due to a number of factors, one of which is the improvement of common software to make it run better on the NVM medium.

REFERENCES

A RESEARCH METHODS

A.1 Part One

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi malesuada, quam in pulvinar varius, metus nunc fermentum urna, id sollicitudin purus odio sit amet enim. Aliquam ullamcorper eu ipsum vel mollis. Curabitur quis dictum nisl. Phasellus vel semper risus, et lacinia dolor. Integer ultricies commodo sem nec semper.

A.2 Part Two

Etiam commodo feugiat nisl pulvinar pellentesque. Etiam auctor sodales ligula, non varius nibh pulvinar semper. Suspendisse nec lectus non ipsum convallis congue hendrerit vitae sapien. Donec at laoreet eros. Vivamus non purus placerat, scelerisque diam eu, cursus ante. Etiam aliquam tortor auctor efficitur mattis.

B ONLINE RESOURCES

Nam id fermentum dui. Suspendisse sagittis tortor a nulla mollis, in pulvinar ex pretium. Sed interdum orci quis metus euismod, et sagittis enim maximus. Vestibulum gravida massa ut felis suscipit congue. Quisque mattis elit a risus ultrices commodo venenatis eget dui. Etiam sagittis eleifend elementum.

Nam interdum magna at lectus dignissim, ac dignissim lorem rhoncus. Maecenas eu arcu ac neque placerat aliquam. Nunc pulvinar massa et mattis lacinia.