Carnegie Mellon University

CARNEGIE INSTITUTE OF TECHNOLOGY

THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF Master of Science

TITLE ______________________________

____

PRESENTED BY ______________________________

ACCEPTED BY THE DEPARTMENT OF

Information Networking Institute

______________________________ ______________
THESIS ADVISOR DATE

______________________________ ______________
ACADEMIC ADVISOR DATE

______________________________ ______________
DEPARTMENT HEAD DATE

APPROVED BY THE COLLEGE COUNCIL

______________________________ ______________
DEAN DATE

Comparing Performance of Different Cleaning Algorithms for SMR disks

Submitted in partial fulfillment of the requirements for

the degree of

Master of Science

in

Information Networking

Mukul Kumar Singh

B.E. Visvesvaraya Technological University

Carnegie Mellon University
Pittsburgh, PA

April, 2014

Copyright © 2014 by Mukul Kumar Singh
All rights reserved except the rights granted by the
Creative Commons Attribution-Noncommercial Licence

Acknowledgements

I would like to thank my thesis advisor Professor Garth Gibson for all the guidance and continuous support. I would like to thank him for developing a research instinct in me. His guidance helped me a lot while I was working on my thesis. I would also like to thank the reader of my thesis Professor Greg Ganger. His comments were invaluable in finishing my thesis. I would also like to thank them for giving me an opportunity to be a part of Parallel Data Laboratory at Carnegie Mellon University.

I would like to thank Seagate for funding this project through the Data Storage Systems Center at CMU. I also thank the members and companies of the PDL Consortium (including Actifio, American Power Conversion, EMC Corporation, Facebook, Fusion-io, Google, Hewlett-Packard Labs, Hitachi, Huawei Technologies Co., Intel Corporation, Microsoft Research, NEC Laboratories, NetApp, Inc. Oracle Corporation, Samsung Information Systems America, Seagate Technology, Symantec Corporation and Western Digital) for their interest, insights, feedback, and support.

I would also like to thank my friends and colleagues at Carnegie Mellon for helping me with discussions and their continuous support. Finally I would like to thank my parents, brother and sister-in-law for their encouragement and support. I would also like to thank my best friends Soumya Koduri and Rahul Goyal for their endless support and enthusiasm.

Abstract

Shingled Magnetic Recording (SMR) promises to sustain current growth in disk drive capacities with minimal change in the current disk drive technology. Shingling implies overlapping of tracks in a hard drive. Shingling would cause overwrites on down-track sectors with each sector write, hence new interfaces are being proposed to allow host software to exploit SMR with minimal change. An obvious interface is a Shingled Translation Layer which is akin to a Flash Translation Layer. Here the disk can completely hide the layer of remapping and background cleaning, but this comes at the cost of complexity in the disk processor and hard-to-predict performance changes. Other interfaces which enable the host application to handle shingling have been proposed as well. In a strict append model , the disk is divided into fixed sized bands and data is written to a particular band in a strict append order, with cleaning done by resetting the write cursor to the beginning of a band. Another promising interface, Caveat Scriptor, gives the host an address space of all possible sectors. In-order to handle shingling, this interface exposes two drive parameters to determine which sectors may or will not be damaged because of a certain write. These parameters are Drive No Overlap Range (DNOR) and Drive Isolation Distance (DID). This paper will explain these parameters, explain the design of a filesystem designed for this extreme interface, caveat scriptor, and compare the cleaning performance of a filesystem designed for the Caveat Scriptor interface to one designed for the Strict Append interface.

Table of Contents

Acknowledgements **ii**

Abstract **iii**

List of Tables **vi**

List of Figures **vii**

1 Introduction **1**

2 Background **5**

2.1 Types of FileSystem : A cleaning perspective 5

2.1.1 Cleaner thread 6

2.1.2 Implicit cleanup 6

2.2 Strict Append Filesystem SMRFs 7

2.2.1 Disk Apis 7

2.2.2 Filesystem 8

2.2.3 Cleaning 8

3 Caveat Scriptor **11**

3.1 Motivation 11

3.2 Definition 11

3.3 Disk Semantics 13

3.4 Support for Implicit Cleaning 13

4 Caveat Scriptor - SMRfs **15**

4.1 Overwrite Detection 16

4.2 Disk Interfaces . . . 16
4.2.1 caveat modsense . . . 16
4.2.2 caveat write . . . 17
4.2.3 caveat read . . . 17
4.2.4 caveat create . . . 17
4.2.5 caveat open . . . 17
4.2.6 caveat dump . . . 17
4.3 Disk Partitioning . . . 18
4.3.1 Unshingled Partition . . . 18
4.3.2 Shingled Partition . . . 18
4.4 File System Semantics . . . 18
4.4.1 File System Metadata . . . 19
4.4.2 File System Workflows . . . 20
4.4.3 Block allocation . . . 22
4.4.4 Allocation Algorithms . . . 23

5 Results 26
5.1 Testing Infrastructure . . . 26
5.2 Performance Test . . . 26
5.2.1 Large File Benchmark . . . 27
5.2.2 Vdbench . . . 30
5.2.3 PostMark . . . 30
5.2.4 Linux Compile . . . 31
5.3 Capacity Test . . . 32
5.3.1 Test with DID as 16 4KB blocks . . . 33
5.3.2 Test with DID as 512 4KB blocks . . . 36

6 Conclusion 39

7 Future Work 41

Bibliography 42

List of Tables

Table 5.1 Fragmentation with DID as 16 4KB blocks 34

Table 5.2 Fragmentation with DID as 512 4KB blocks 36

List of Figures

Figure 2.1 Strict Append SMRfs Workflow 9

Figure 3.1 DID and DNOR calculation 12

Figure 4.1 Block extent freelist 20
Figure 4.2 Caveat SMR Workflows 21
Figure 4.3 Write Dataflow 22
Figure 4.4 Extent segregated list 24

Figure 5.1 Read throughput for LFS test with 100MB file 28
Figure 5.2 Write throughput for LFS test with 100MB file 28
Figure 5.3 Read throughput for LFS test with 500MB file 29
Figure 5.4 Write throughput for LFS test with 500MB file 29
Figure 5.5 File operations per second for vdbench benchmark 30
Figure 5.6 File operations per second for Postmark benchmark 31
Figure 5.7 Linux Compilation time in seconds 32
Figure 5.8 Percentage of unusable free blocks over total blocks in the disk 34
Figure 5.9 Overallocation & cleaning performance for DID as 16 4KB blocks 35
Figure 5.10 Percentage of unusable free blocks over total blocks in the disk 37
Figure 5.11 Overallocation & cleaning performance for DID as 512 4KB blocks 38

Abbreviations

SMR	Shingled Magnetic Recording, a new disk drive technology in which the tracks are organised as shingles.
DID	Drive Isolation Distance, the maximum number of downstream sectors which may be overwritten because of a write on a particular sector on a SMR drive.
DNOR	Drive No Overlap Range, the minimum number of sectors downstream which will definitely not be overwritten because of write on a particular sector in a SMR drive.
KB, MB, GB	These are the units of storage capacities in multiple of bytes. These units are Kilobyte($2\hat{1}0$), Megabyte($2\hat{2}0$) and Gigabyte($2\hat{3}0$) respectively.
LBA	Sectors in a disk are addressed by logical block address aka lba.

1

Introduction

Hard disk drives have been in use for a significant amount of time now. Even with the availability of Solid State Drives(SSDs), hard disk drives are still popular because they provides cheap and reliable storage. Moore's Law states that the speed of the processor double every 18 months. But according to Mark Kryder, when we compare this growth rate to the rate of growth of hard disk drive capacities, even that rate seems slow[25]. This explosive growth in the disk drive capacities has been possible with the support of new disk drive technologies as perpendicular recording etc[14].

The growth which is being currently observed is reaching its limit in terms of areal densities. Sustained growth at this rate is not possible without new technologies and significant changes to the hard disk drive technologies and interfaces. Most of proposed technologies are disruptive and define new methods to store data in a denser format on the hard disk drive then what is possible using current hard disk drive technologies. More disruptive the technology, the longer it will take for it to be assimilated in the market, also parameters such as performance, cost and compatibility with the current hardware and software architecture will also play a very important factor in their acceptance in the market. Technologies such as Bit

Patterned Magnetic Recording (BPMR)[13] and Heat Assisted Magnetic recording (HAMR)[22] promise significant increase in the areal density but they will also require significant change in the drive physical layout, interfaces, architecture and operating techniques, hence it might take a considerable amount of time for these technologies to be accepted in the market[19].

Shingled Magnetic Recording(SMR)[15, 12, 17] promises to increase the areal densities of the hard drive without significant changes in either the manufacturing process or the operation techniques of these hard drives. SMR will be able to support the current growth rate being observed with the hard drives areal densities.

One of the biggest factors hindering the increase in areal densities is the disparity in the width of read and the write head. In order to provide persistent data storage, data needs to be written on a wider pitch to make sure that enough magnetic flux is generated to store the data, however data can be read correctly from a much narrower pitch than what it is written to. Write heads can't be made any narrower than what they currently are, it will be unable to maintain sufficient magnetic flux to store the data correctly[24]. Currently track pitch is defined by the width of the write head and this ensures that two adjacent tracks do not overlap. Read heads are also aligned to write pitch as well and it can read data correctly from one track at a time. Although write heads can't be made narrower than what they are, read heads still can be, hence new methods for arranging tracks can be designed to increase areal densities.

SMR works by overlapping the track in a hard drive in a method similar to shingles on the roof of a house. Hence a track on SMR drive overlap with 'n' other downstream tracks. Such an arrangement will help in increasing the areal densities for these drives. Though the data can be read correctly using a narrower read head, writing data on any random sector will result in extremely peculiar write anomalies. A write on any group of sectors might overwrite data on certain downstream sector.

The extent of overwrite is influenced by various disk geometry parameters such as position of the sector being written, level of shingling supported on the hard drive, zoning etc. Hence new methods to handle this write anomaly needs to be designed, however data can still be read conventionally using a narrower read head. Hence SMR promises to offer increased areal densities with minimal changes in the current disk technology.

Various methods to make use of SMR drives have been discussed in Feldman13 and can be broadly classifies as Drive Managed SMR, Host Assistive SMR and Co-operative SMR[14].

Drive Managed SMR describes methods in which the disk firmware manages the write anomalies of the drives. This can be done by maintaining a mapping of the logical block numbers (LBAs) to the sectors on the hard drive. A write which can cause an overwrite on downstream sectors can be redirected to another set of sectors in the drive and an entry for this can be maintained in the mapping table. This method is similar to Flash Translation Layer(FTL) which is currently being used extensively on Solid State Drives(SSDs). Such a method will not require any changes in the current file system implementation and can be an important factor for the early adoption of this drive technology. However with such an approach, file system developers will not be able to ensure good locality of the data on the drive and on a drive with mechanical parts, locality will play a very important role in defining the performance characteristics of the drive[15].

Host Assisted SMR is another approach in which, it is the responsibility of the host application to manage the write anomalies of the drive. Such a method will either require changes in the current filesystem implementations or new file systems need to be developed to leverage these drives. By letting the host application manage the write anomaly, host applications will be able to guarantee good locality for the data stored on these drives and hence provide good performance characteristics for

the application. As discussed in Feldman13, these disks can be modelled as either as Strict Append SMR[14] or as Caveat Scriptor SMR[14]. Strict Append model visualizes the drive as being divided into multiple bands, where each band is of fixed capacity. Two bands are separated by an 'interband gap', with such an arrangment, writes on one band do not overwrite data on the other band and hence localize the shingling effect to only one band[12]. On the other hand in the Caveat Scriptor model, entire drive is exposed to the application, and it is responsibility of the application to cope up with the characteristics of the drive.

2

Background

This chapter describes the motivation behind Caveat Scriptor.

2.1 Types of FileSystem : A cleaning perspective

Although filesystems can be classified into many different types depending upon its applications, use cases, devices being supported, optimization and performance etc, here we will try to classify them on the type of allocation, de allocation and cleaning algorithm used with the filesystems.

All the filesystems need to keep track of free and allocated data blocks, allocated data blocks are one which have live data which the user is accessing on them while free blocks are one which do not have live data. Blocks are marked live when user writes data on a block and marked free when file is deleted. Depending upon the allocation algorithm, a background cleaner thread might be used to perform garbage collection on the file system otherwise cleaning can be done implicitly as well.

2.1.1 Cleaner thread

If we look into log structured file systems[21] like BSD-LFS[20] and NILFS[7] which have been designed to improve the write performance, require a garbage collection algorithm to free blocks. This cleaning algorithm moves live data around in the filesystem to free up segments. These freed segments can be used later to write data. Such an implementation adds the complexity of managing a cleaner thread. This thread identifies live and dead blocks and copies data into a new segment to free data blocks.

2.1.2 Implicit cleanup

On the other hand, there are filesytems which perform implicit garbage collection, like the ext 2/3/4 filesystems. These filesystems allocate blocks when a file or directory is created from the block bitmap. Blocks in this bitmap are marked as allocated when blocks are allocated for a write operations and free when these blocks are no longer in use[1]. This bitmap is used by the allocation mechanism to satisfy the next allocation request. Using such a method performs implicit garbage collection because the block allocation are done from the bitmap and no cleaner thread is used to reclaim blocks later, blocks are marked as free as soon as they are not in use.

These methods for allocating and deallocating data blocks have been developed assuming the current behaviour of disk drives. Currently the effects of write onto a particular sector are localized to the sector being written to. This approach will not work on SMR drives, as a write will overwrite data on downstream sectors. Hence new allocation, de-allocation and cleaning mechanism needs to be developed to support shingled drives.

2.2 Strict Append Filesystem SMRFs

In Strict Append, a SMR disk is modelled as a drive which is divided into multiple bands, where each band is of a fixed capacity. Each band will also maintain a write pointer to which the new data is appended to. When a band does not contain any live data then the write pointer for this band can be reset to the start of the band and new data can then be written using the append only semantic[17].This model will also require new disk interfaces to add support for bands and the write pointer in the band.

2.2.1 Disk Apis

modesense

This command is used to determine the characteristics of the hard drive such as the sector size, band size and the number of bands in the drive.

managebands

This command will help in managing the band inside the disk. This command will also provide interfaces to set the position of the write pointer for a band, set the position of the write pointer for a band and also a method to reset the pointer for all the bands.

read

This command is similar to the read command on a normal drive, this command will also provide method to specify the band id and also the relative band address inside this band from where the data would be read.

write

This command will also take a parameter to specify the band id where the data would be written to. The data to be written will be appended after the position of

the current write pointer. Once the data has been written, then the write pointer for the band will be updated to the new position.

2.2.2 Filesystem

As explained in Suresh's report Strict Append-SMRfs[23] is a filesystem which is implemented on strict append SMR disk. Strict Append-SMRfs is a FUSE based filesystem and it also had a 2GB buffercache. All the read and write operations cause the file to be staged in the buffercache. When the buffercache decides to move the file to the disk then it identifies a band to be written to using the band meminfo module. This module keeps information about the free capacity of the various bands in the drive. Once a band is identified then the data is written on the disk. The drive is partitioned into unshingled and shingled partition. Unshingled partition is used to store the file metadata while the shingled partition is used to store the actual file data. A file on Strict Append-SMRfs resides only in one band. A file is identified by a pointer which contains the band id, the offset inside the band where the file data resides and number of blocks of the file which are in that band. Buffercache in SMRfs supports full files and all the writes for a file are collated in the buffercaches before the file is pushed to the disk when it is closed. When a file is overwritten, then the file is moved to a new location and the blocks which were previously allocated are marked for cleaning. Since the data is written inside a band in a strictly append manner hence old live data should be moved to a new band to free up space in a partially filled up band.

2.2.3 Cleaning

When the filesystem reaches a certain pre-determined threshold on the filled up filesystem capacity or when cleaning is explicitly triggered, a garbage collection thread runs to free up bands which contain partial live data. Cleaning works by

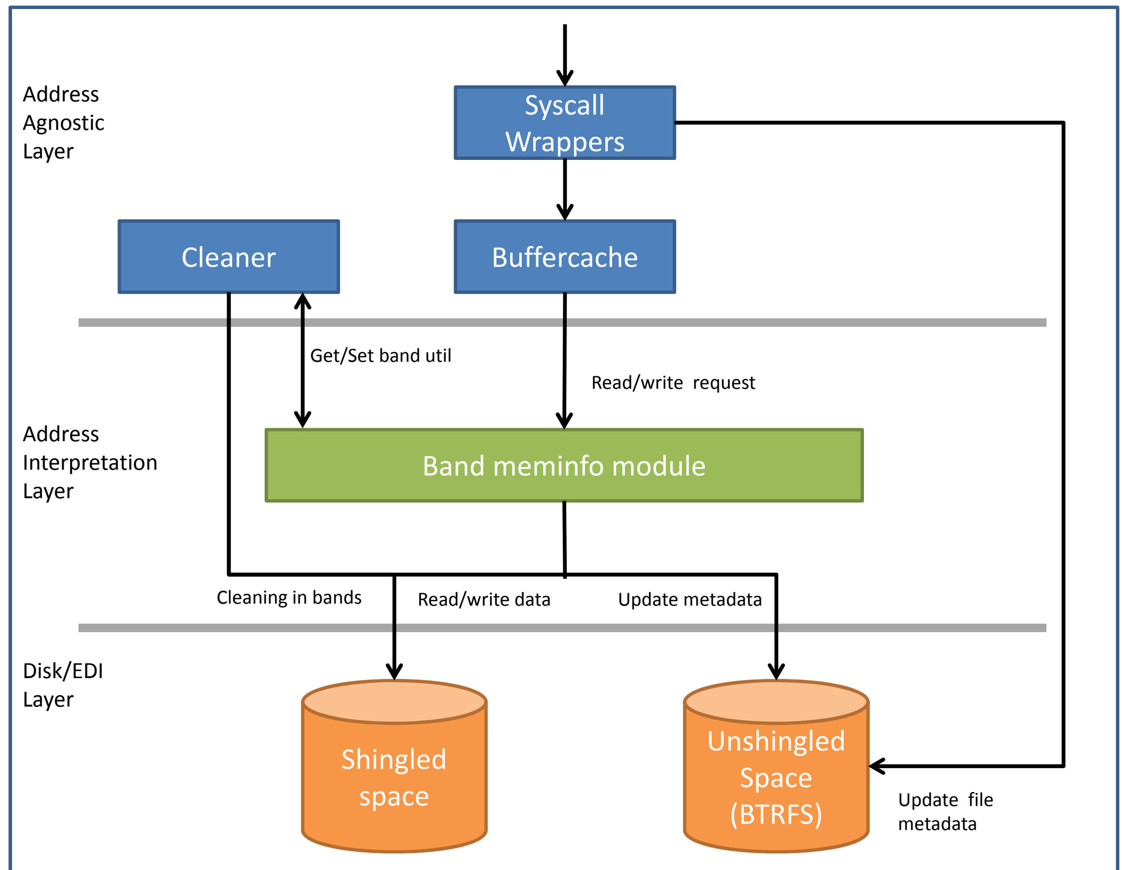

Figure 2.1: Strict Append SMRfs Workflow

moving the live data in one band to another band and then resetting position of the write pointer in this band to the start of the band, hence cleaning the band and freeing up the dead blocks in the band. This thread then updates the inode structure accordingly to the new location of the file. Cleaning is influenced by various cleaning strategies which have different efficiencies and cleaning cost associated with them. Cleaning cost is attributed to the compute overhead spent and disk resources used while performing the cleaning. It is also influenced by other parameters as the

thread management overhead and inode update overhead etc. Such a filesystem will also require running a cleaning algorithm even when the dataset has lots of small files which have a small life span.

3

Caveat Scriptor

3.1 Motivation

As observed earlier, strict append approach to SMR helps in avoiding the write anomaly of the caveat drive, but it introduces the cost of running a cleaning algorithm on the filesystem to free up the blocks, hence attributing to additional complexity and cost.

In order to improve the cleaning cost observed with Strict Append SMRfs, Caveat Scriptor is proposed as a model which will expose important disk parameters to the host application, which will have to take care of the write anomaly of the disk. Such a model will also help in performing implicit garbage collection in the file system. While this might add extra complexity in the allocation algorithm and filesystem workflows but will ensure a much lower cleaning cost associated with the filesystem.

3.2 Definition

Caveat Scriptor models the SMR disk as a standard disk, which is addressed using logical block number, where size of sector is configurable (4KB in this document).

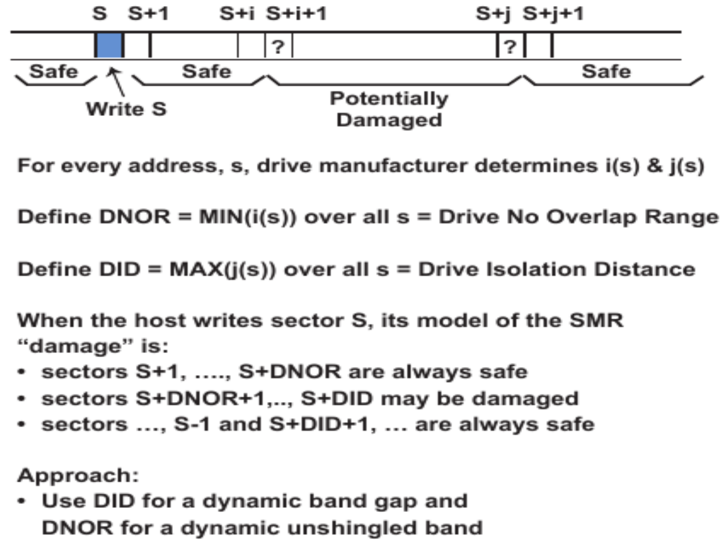

Figure 3.1: DID and DNOR calculation

In order to describe the unique nature of these drives, minor disk interface changes are necessary. These changes will help in differentiating these drives from the regular hard disk drives.

Caveat disks introduce two important parameters called the Drive No Overlap Range(DNOR)[14] and the Drive Isolation Distance(DID)[14]. Drive No Overlap Range is the minimum number of sectors which are adjacent to a particular sector and will not be damaged by a write on that sector. Drive Isolation Distance is the maximum number of downstream sectors which might be overwritten because of write on a particular sector. These parameters are heavily influenced by the actual disk geometry: number of tracks which are overlapping, disk layout, number of platters, number of sector on each track and zoning. The disk drive can be conservative with these parameters, but it must guarantee that DNOR and DID are valid for every write at any address on the drive. These parameters are explained in Figure 3.1.

DNOR for a disk with variable number of sectors per track is defined as minimum

of all DNORs for every track while DID for a disk is maximum of all the DIDs for each track. DID and DNOR must be selected so that they will work for all the addresses, even if the disk has zoning as well.

The parameters DID and DNOR satisfy the equation 0 <= DNOR <= DID. Whenever a write is performed on sectors from i to j, then sectors from MAX(i + 1 + DNOR, j + 1) to j + DID might be damaged. Number of sectors which actually get damaged depend a lot on the sector number, position of the sector on the track and alignment of the sector with adjacent tracks.

Sectors which will actually be damaged because of a write on a particular sector can be determined correctly with the complete knowledge of the disk geometry, but that would require keeping an extremely large metadata to identify the precise sectors which are definitely damaged because of a particular write. With the help of these two parameters, sectors which might be damaged because of the write can easily be determined. These parameters will simplify the calculation for the "maybe" damaged sectors.

3.3 Disk Semantics

A Caveat Scriptor disk is defined by four parameters which are disk size, sector size, DID and DNOR. Support should be added to disk interface protocols to fetch these parameters from the disk.

3.4 Support for Implicit Cleaning

When Caveat Scriptor model is compared to Strict Append in which cleaning was required because a band contained both live and dead data and since the data could only be written in a strict append manner a band should be cleaned before it can be re written again. In order to clean a band, the current live data was moved to a different band, the write pointer for the band reset to the start of the band and thats

when the band can be written again. With the Caveat Scriptor model, the data can be written anywhere on the disk ensuring that the shingling effects of the writes are taken care of. With this model implicit cleaning can be supported by maintaining a bitmap of free and allocated blocks and using that bitmap in making allocation decisions while allocating the blocks for write.

Filesystem on the caveat disk can make use of this bitmap and pad the write with extra unused sectors which will be used to absorb the shingled effects of the write operation. These extra padded sectors can be reclaimed later and used for the subsequent allocation decisions.

4

Caveat Scriptor - SMRfs

In this chapter, we describe the disk interface for Caveat Scriptor. We also describe a method to implement filesystem over Caveat Scriptor disk.

We implemented a host assisted file system in which the filesystem handles the write anomalies of the Caveat Scriptor disks. The motivation was to build a filesystem on a Caveat Scriptor disk which perform cleaning implicitly. Such a method will help in efficient use of SMR disks with minimal cleaning overhead. This filesystem is developed using the File System in UserSpace(FUSE), this makes the development of the filesystem easier. Also this file system has been developed by modifying the Strict Append-SMRfs, metadata and buffercache implementation in Strict Append-SMRfs has been used in this implementation. This helped in minimizing the development time.

This implementation will make use of the DNOR and DID parameters exposed by the disk and use them while making allocation decisions and also while writing data on to the disk.

4.1 Overwrite Detection

Since the data on a downstream sector can be overwritten because of write on an upstream sector, method to detect read from an overwritten sector should be added to the disk firmware. This method will help in identifying a read on sector which has previously been damaged by an overwrite. To support such a method, a bitmap should be maintained by the caveat disk. This bitmap keeps a track of overwrite on the disk blocks. This bitmap will ensure that a read request from a "maybe damaged" sector fails and appropriate error is returned. Initially all the sectors in the disk are marked as maybe damaged. Whenever a write is performed on any set of sectors, they are marked as "not damaged" while downstream overwritten sectors derived from DID and DNOR are marked as maybe damaged. Hence after writes on the disk drive, "not damaged" sectors will contain user data while "maybe damaged" sectors have been overwritten because of write on other sectors.

Such a method would help in identifying and debugging invalid read requests to the disk.

4.2 Disk Interfaces

Caveat disk apis are the api for the software layer wrapped around the disk. These apis have been designed such that the interfaces are independent of the filesystem implemented over these disks.

4.2.1 caveat modsense

Caveat modsense will fetch important drive parameters from the disk, it will fetch parameters as sector size, disk size and the values of DID and DNOR. These parameters will greatly influence the behaviour of the disk and also the filesystem designed over these disks.

4.2.2 caveat write

Caveat write will accept a buffer, the starting logical block address on the disk and number of sector to be written. This api will write the data on the disk and will also determine the number of sectors which might potentially get damaged because of this write and mark them appropriately in the caveat bitmap.

4.2.3 caveat read

Caveat read will accept a buffer, the starting logical block address on the disk and number of sector to be read. This api will first check that the sectors being read are not damaged from a previous write using the disk bitmap and then will read the required number of sectors from the disk. If an error is detected, then read fails and an appropriate error is returned.

4.2.4 caveat create

This api is used to create an emulated caveat disk. This will create and initialize the disk with the disk parameters as sector size, disk size, DID and DNOR. This will also initialize disk bitmap as maybe damaged for all sectors.

4.2.5 caveat open

Caveat open opens a disk which was previously created with caveat create api and initializes the in-memory data structures. This disk can then be accessed using caveat_read and caveat_write apis.

4.2.6 caveat dump

This api is present only for debugging purposes and can be used to dump caveat disk information when an error occurs. This api was found to be extremely useful while developing the filesystem for caveat disk.

4.3 Disk Partitioning

Caveat drive is divided into two partitions, one of the partition is unshingled, which implies that all the write operations on the blocks of this partition do not cause any shingling effect while the other partition still exhibits shingling nature of the drive.

4.3.1 Unshingled Partition

A small portion of the drive is marked as unshingled where all the file metadata is stored, other information such as directory and extended attributes are also stored in this partition. This partition is formatted using ext4 filesystem. This method was used to expedite the prototyping. With correct implementation all the metadata can also be stored in the unshingled portion as well, but would require longer development cycle. Unshingled portion can be generated in shingled area by appending DNOR sectors by DID number of sectors. This will ensure that the writes on these unshingled sectors do not cause any overwriting on downstream sectors.

4.3.2 Shingled Partition

Shingled partition is used to store the actual file data blocks. Writes on this portion is susceptible to write anomalies of shingled drives.

4.4 File System Semantics

Caveat Scriptor-SMRfs implements a 2GB buffercache. Currently the buffercache supports full files only. All the read and write to a file will cause the entire file to be read to the buffercache. When a file is closed and is pushed onto the disk, even then the entire file is stored contiguously on the disk

4.4.1 File System Metadata

File System Bitmap

Caveat SMRfs maintains a filesystem bitmap, this bitmap is used to make allocation decisions. These allocation decisions are triggered by a write operation on the filesystem. It should be remembered that this bitmap is different from the bitmap maintained by the caveat disk. This bitmap is used to determine the free block in the filesystem. This bitmap is updated everytime a file is stored by the filesystem onto the disk. Blocks will be marked as allocated in the bitmap when the blocks are allocated for write operation on a file while they will be marked as free when a file is unlinked.

Initially when the filesystem is created for the first time, all the blocks are marked as free, subsequent write and unlink operation will change the bitmap.

Block freelist

Blocks which are marked as free in the bitmap are also tracked in a linked list as shown in figure Figure 4.1. Every node in this linked list tracks an extent of blocks. An extent is defined by the starting block address and number of contiguous blocks from the starting block. This linked list is sorted in ascending order on logical block addresses, sorting will help in identifying the adjacent free blocks. This will help in identifying extents which can be coalesced together to create a bigger extent. All the allocation and de-allocation requests will be done with the help of this linked list and will also modify the list as well.

Such a method to store this list in memory will perform better than lookups to block bitmap which is stored on the disk and would require frequent disk operations.

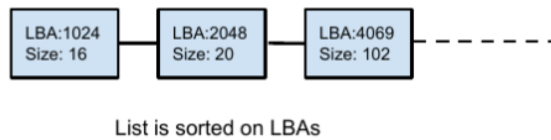

Figure 4.1: Block extent freelist

File System Inode

Since SMRfs stores a file contiguously on the disk, an inode contains the starting block address on the disk where the file is stored and its length.

4.4.2 File System Workflows

All the filesystem operation can be divided into 2 sets, (1) operations on metadata and (2) operations on data blocks. Workflows which require access to the file data are depicted in Figure 4.2 and explained in following section.

init

This api is called when the filsystem is mounted. This api will read the filesystem bitmap and will determine the free and allocated blocks and will construct the freelist as well.

write

This api will write data on the shingled portion of the disk. Whenever the buffercache decides to push a file to the disk, this api ensures that live data on the drive is not overwritten. It determines the number of blocks which are to be written to the disk and then appends DID number of blocks as footer to ensure that overwrite caused

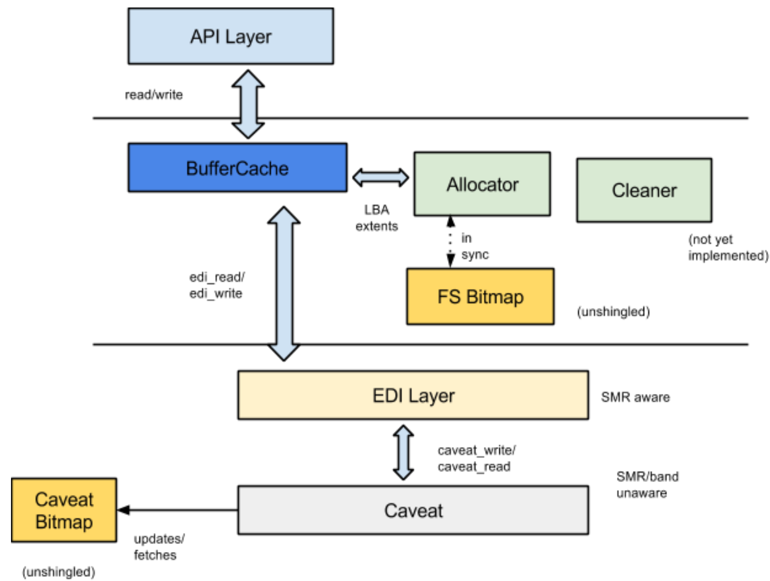

Figure 4.2: Caveat SMR Workflows

by this write does not overwrites other files, this is shown in Figure 4.3. This footer is only appended when the blocks for this file are being allocated on the disk and is freed up when the write to the disk succeeds. A write operation will cause modification to the filesystem and disk bitmap as well.

It should be noted that any inplace writes for a file, causes the file to be relocated to another set of blocks on the disk. The space reserved by the file earlier is freed and inode is updated with the location of the file.

read

Read operation will not be affected by the shingling nature of the drive. Read operations can be satisfied by the file information stored in the inode structure i.e. starting block address and the size of the file.

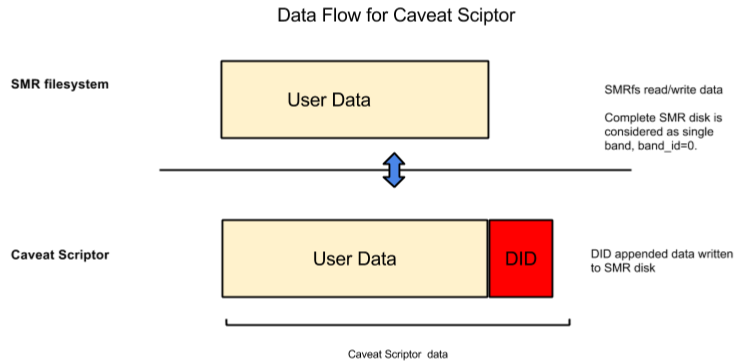

Figure 4.3: Write Dataflow

unlink

Whenever a file is deleted, and if the number of links are zero then the blocks allocated to the file can be marked as free. Blocks will be updated both in the filesystem bitmap as well as in the freelist. While updating the blocks in the freelist, blocks adjacent to a freelist extent will also be checked in the bitmap. If the adjacent blocks are found to be free then these extents will be coalesced to create a bigger extent. These blocks will be used later for other allocation decisions.

truncate

Truncate will be similar to unlink and will free the blocks in the file if the file is larger than the offset specified in truncate, otherwise truncate might lead to allocation of blocks as well.

4.4.3 Block allocation

All the free blocks in the file system are clustered into extents, where each extent maps contiguous blocks in the drive. These extents are sorted on block addresses and referenced using a linked list. This list is called as freelist as explained earlier.

All the allocation and deallocation operations will access and modify this freelist.

These operations will cause fragmentation in the file system. After sufficient number of operations, there will be extents which have free blocks but can't be used to make allocations because they are too small for the current allocation request. Extents with number of blocks less than DID will not be used for making allocation decisions unless blocks adjacent to it are freed and then these extents are coalesced to create a bigger extent. As discussed earlier use of extents and coalescing will perform implicit cleaning in case of Caveat Scriptor.

4.4.4 Allocation Algorithms

Various allocation algorithms have been explored to determine the effects of allocation strategies on fragmentation. Allocation algorithms also play an important factor in the amount of seek resulting from the operations on the caveat disk.

Best Fit Allocation

In this allocation policy, all the free extents are organised in a segregated list Figure 4.4, in this segregated list extents are grouped together by their sizes. Every group in this list track atleast 1MB of extents and the sizes of these groups grow exponentially in powers of 2. Free nodes inside every group are sorted in increasing order of sizes. Sorting of extents help in best fit and worst fit allocation algorithm.

In this algorithm, allocation decisions are taken with the help of this segregated list. For a given allocation request, a group is selected depending upon the size of the request and then the list is traversed to select an extent with the best fit. When a request to free a certain number of blocks on the disk is made, the required number of blocks are freed and then coalesced. This extent is then inserted into the freelist and the segregated list.

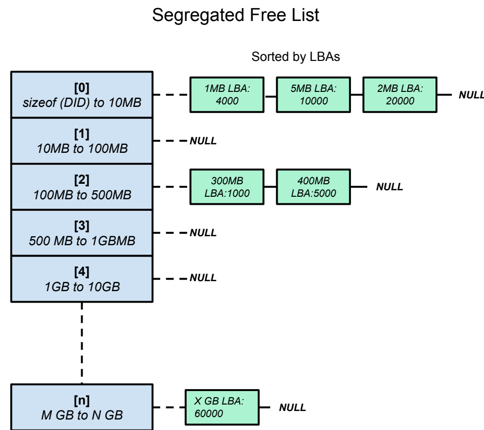

Figure 4.4: Extent segregated list

Worst Fit Allocation

In this allocation policy, all the free extents are organised in a segregated list, similar to the method used in case of best fit allocation. Allocation decision are again done from a group determined by the size of the allocation. However in case of worst fit, allocation is always done from the last free node in the group, this last node is always the biggest node. Free request are again satisfied the same way as in case of best fit allocation policy, free nodes are coalesced before they are added into the freelist and segregated list.

First Fit

In this allocation policy, all allocation decision are made using the freelist in which all the free nodes are sorted in increasing order of logical block address. For every allocation request the freelist is searched from the first node and required number of blocks are allocated from the first free node with sufficient number of blocks. A

request to free the blocks works in exactly the same way as defined for the previous algorithms.

Next Fit

This allocation policy works on the allocation requests from the freelist and the allocator keeps a track of the last allocation request and performs the next allocation from an extent which adjacently after the previously allocated lba. If the previously allocated lba is in between a extent and the extent has sufficient number of blocks then, the extent will be split into two. Allocation is satisfied from one extent while the other is added back into the freelist. A request to free the blocks works in exactly the same way as defined for the previous algorithms.

Last Accessed Postion Fit

In this allocation policy too, the allocation requests are satisfied from the freelist. The allocator keep the track of the last disk access operation. This operation can either be a read or write. Allocator queries the disk for the current location of the disk head. In order to fetch this location, support was added into the disk emulation api to return the current location of the disk head. This location is then used to perform the allocation from the freelist using an extent whose whose starting block address adjacently after the queried position. Also similar to next fit, if the previously allocated lba is in between a extent and the extent has sufficient number of blocks then, the extent will be split into two. Allocation is satisfied from one extent while the other is added back into the freelist. A request to free the blocks works in exactly the same way as defined for the previous algorithms.

5

Results

In this chapter we will describe the testing infrastructure and discuss the test results in detail as well.

5.1 Testing Infrastructure

All the test were performed on NSF Probe Cluster [16, 26, 8] nodes available with Parallel Data Lab at Carnegie Mellon University. Every node in the Marmot has single core AMD Opetron 1Ghz processor, 8 GB RAM and a Western Digital SATA 7200 rpm 2TB WD Caviar Black hard drive.

Caveat drives have been emulated over the standard hard drives with appropriate DID and DNOR values. Different values of DID and DNOR have been tried while running the capacity test.

5.2 Performance Test

Performance of the Caveat Scriptor SMRfs was compared with ext4[3], NILFS[7] and Strict Append-SMRfs. Ext4 is a journaling file system for linux which supports extents and performs cleaning implicitly and is currently the default filesystem shipped

with linux. NILFS is a log structured based file system which supports continuous snapshotting. Strict Append-SMRfs is a file system developed for Strict Append SMR disks, this file system considers the disk to be divided into multiple bands, each band maintains a write pointer where the data to be written is appended to, any data is written on a particular band in a strict append order.

All the performance analysis for Caveat Scriptor was done using the Best-Fit allocation algorithm. Also all the test have been run with size of buffercache limited to 2GB. Performance was compared with the following benchmarks PostMark[18], Lfs and vdbench[9] benchmarks. Apart from these benchmarks, compilation time of Linux code was also measured for each of these filesystems.

While running the performance tests, the value for DNOR is 6 blocks of 4096 / 48 sectors of 512 bytes each and DID is 16 blocks of 4096/ 128 sectors of 512 bytes each. These values are based on the assumption that there are 63 sectors of 512 bytes each on a single track of the Hard drive. These values were derived from the "fdisk -l" output of the disk mentioned above.

5.2.1 Large File Benchmark

LFS test will create files of size 100MB and 500MB each and will run sequential read and write as well as random read and writes on the same file. Throughput is being compared for this benchmark.

When the throughput observed in this benchmark is compared for different filesystems, from Figure 5.1 and Figure 5.3 it can be observed that Caveat Scriptor-SMRfs exhibits similar throughput characteristics as other filesystems. It should be noted however that the throughput for write operation from Figure 5.2 and Figure 5.4, is significantly lower in case of Caveat Scriptor-SMRfs. One of the most probable reason for this can be user space implementation of the filesystem.

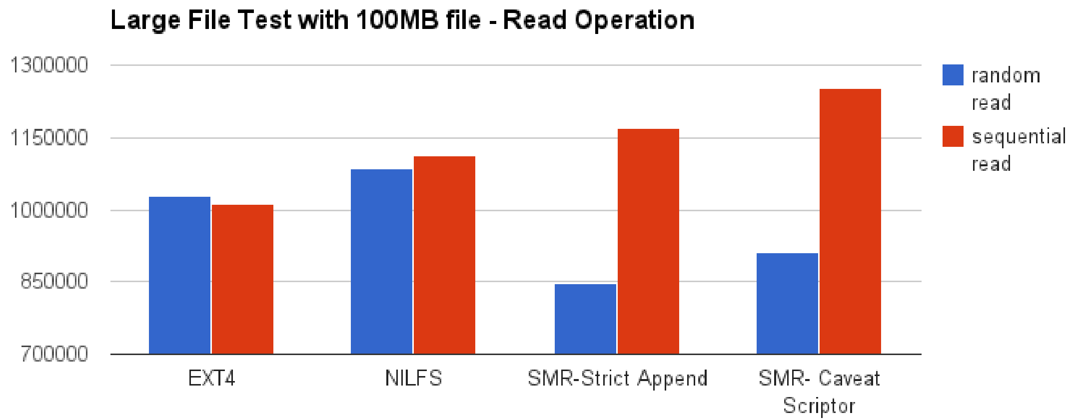

Figure 5.1: Read throughput for LFS test with 100MB file

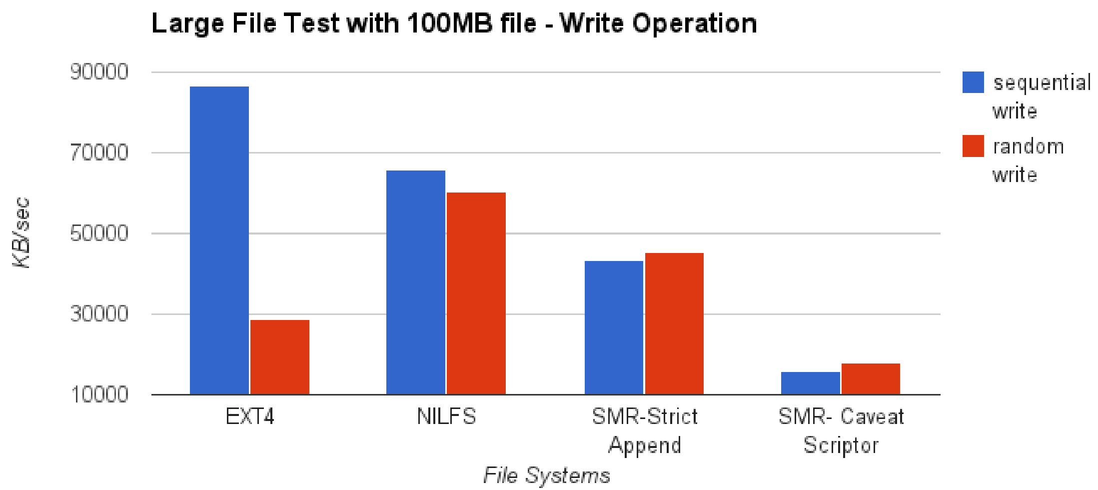

Figure 5.2: Write throughput for LFS test with 100MB file

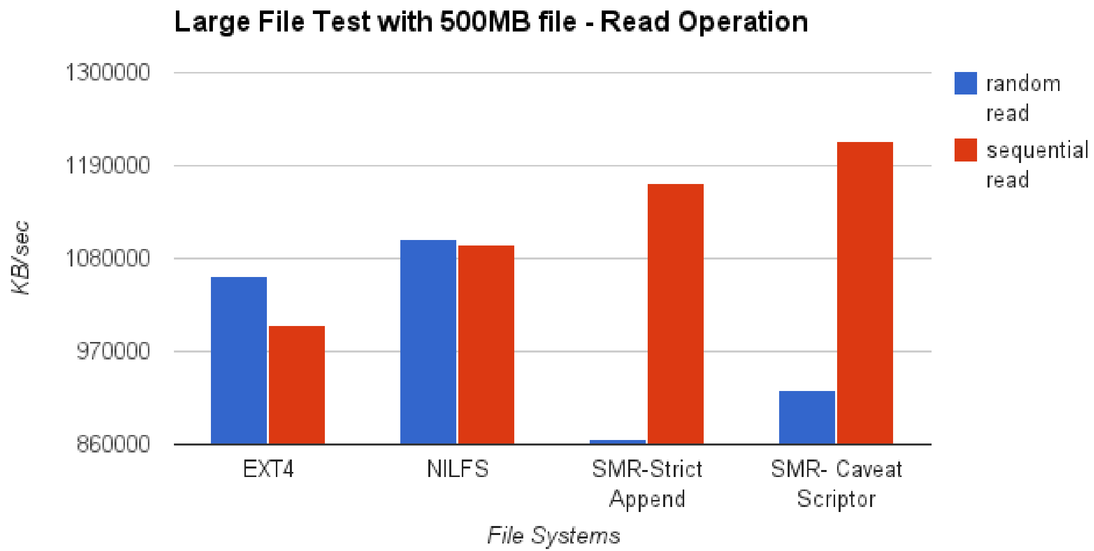

Figure 5.3: Read throughput for LFS test with 500MB file

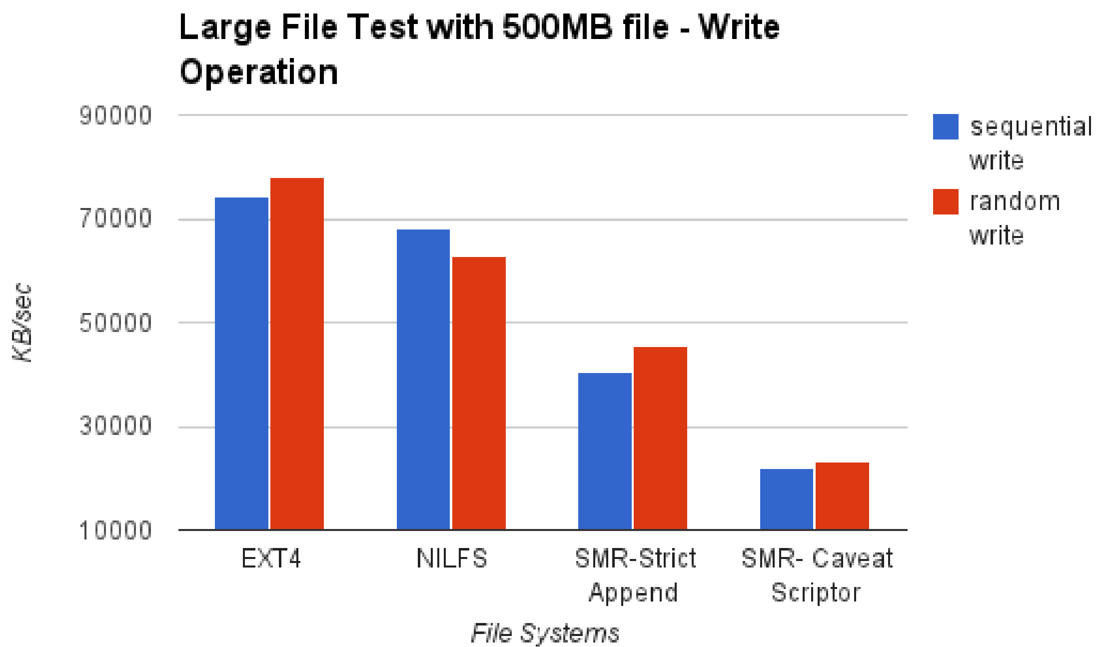

Figure 5.4: Write throughput for LFS test with 500MB file

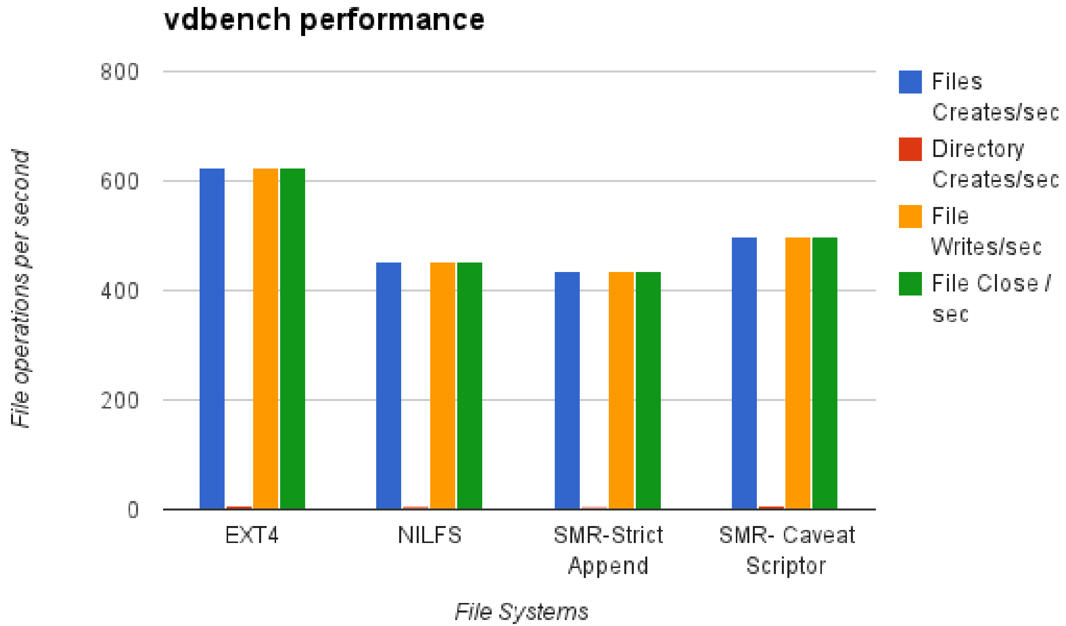

Figure 5.5: File operations per second for vdbench benchmark

5.2.2 Vdbench

Vdbench will create 110 directories and a total of 10,000 small files distributed evenly across these directories. The size of all the files combined is equal to 1.2 GB. This benchmark computes the number of file operations every second.

In case of Vdbench, as shown in figure Figure 5.5, the number of file operation per second are again similar to the other filesystems. Caveat Scriptor SMRfs here performs better than NILFS and Strict Append-SMRfs.

5.2.3 PostMark

Postmark is a benchmark which simulates the behaviour of mail servers. This benchmark will create multiple small files.

As can be seen from Figure 5.6 All the filesystems perform equally in all param-

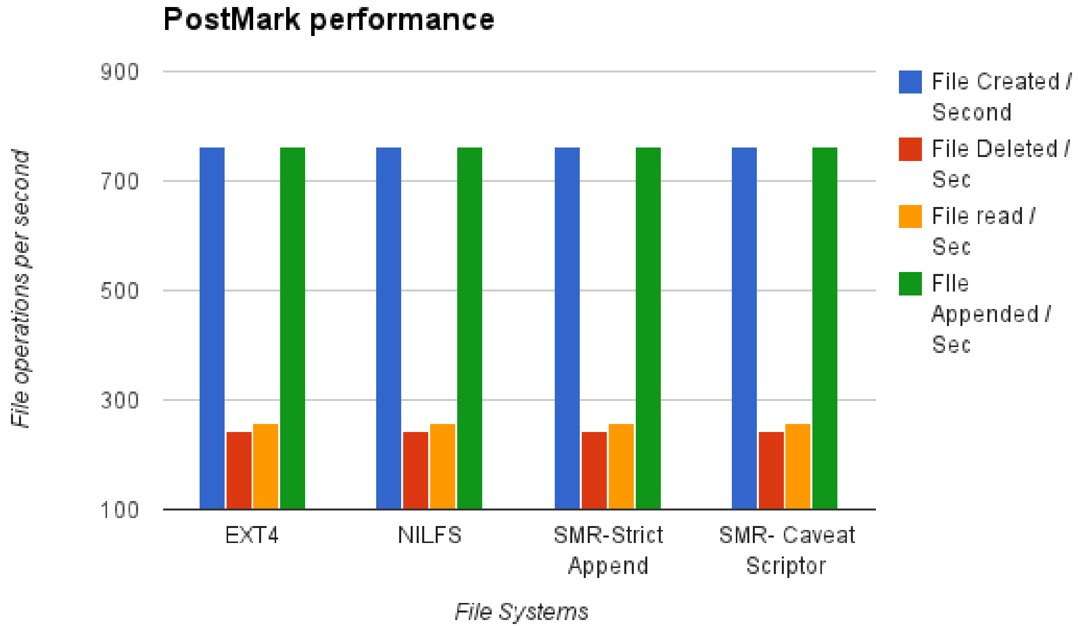

Figure 5.6: File operations per second for Postmark benchmark

eters for this benchmark. The most probable reason for this might be the small size of this benchmark.

5.2.4 Linux Compile

In this test, linux code was compiled on all the four filesystems. For this benchmark linux version 2.6.31.10 was used.

As can be seen from Figure 5.7, Ext4 and Nilfs performs equally good when we compare the compilation time. The same can also be observed for FUSE based Strict Append and Caveat Scriptor as well.

All the performance test result show that Caveat Scriptor-SMRfs exhibits similar performance when compared to Strict Append-SMRfs.

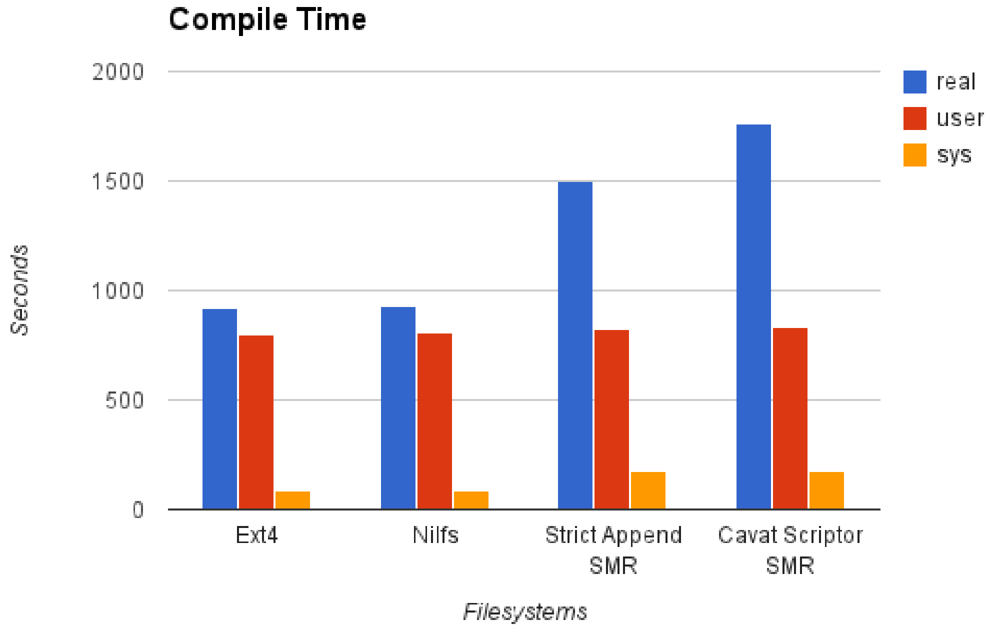

Figure 5.7: Linux Compilation time in seconds

5.3 Capacity Test

In order to evaluate the cleaning performance of the disk, we used Aging tool [27] to create multiple files on the filesystem. This tool takes the filesystem size to fill up as an argument, and then it creates multiple files to reach the required size and then keeps deleting and re-creating new files to maintain the filled-up size of the filesystem. This tool is extremely useful in determining the cleaning efficiency of the file system.

Aging tool can be configured to create different kinds of workloads, here in the current experiment, aging tool will run for 100,000 operations. Operations are random in order and will create and delete files randomly. The ratio of create operations to delete operations is 4:1 until steady state is reached. This tool generates different files determined by a weight associated with each file size. Heaviest weight was

assigned to small files of size 4 KB and 8 KB, this tool will also generate big file of size close to 4 MB a lot less frequently. The files thus created are also deleted in a similar manner. Frequent creation and deletion of files causes aging in the filesystem and will cause frequent coalescing to happen when files will be deleted. This tools hence helps in understanding the cleaning performance of the filesystem.

All the tests have been run for a filesystem of usable capacity of 1 GB, this size is only the size of the data blocks in the filesystem and not the metadata.

Since Caveat Scriptor-SMRfs appends a DID at the end of every write, hence it introduces fragmentation in the filesystem. Also since DID is appended to the end of every write, hence we will never be able to run a benchmark to completion on Caveat Scriptor-SMRfs without overallocating the drive capacity. Hence the aging tool was run multiple times to determine the percentage of overallocation for various allocation algorithms.

In order to check for different DID values, aging tool was run once with DID as 16 blocks of 4KB and then with DID as 512 blocks of 4KB. Both of these test were run on a filesystem with usable capacity of 1GB. Different over-allocation percentages of the disk capacities were observed for both the values of DID.

5.3.1 Test with DID as 16 4KB blocks

For this value of DID it can be observed from Figure 5.8 and Table 5.1 that different values of overallocation were needed to run a 1GB workload on this filesystem. The lowest value of overallocation(2%) was observed for Next fit and Last Accessed Position fit algorithm while the highest overallocation(5%) was needed for Worst fit and Best Fit allocation algorithm.

If we also look into Figure 5.9, it is observed that the total amount of blocks allocated by the filesystem is 2.5 times the usable filesystem capacity of 1GB. The implicit cleaning algorithm used in Caveat Scriptor-SMRfs marks the blocks as free

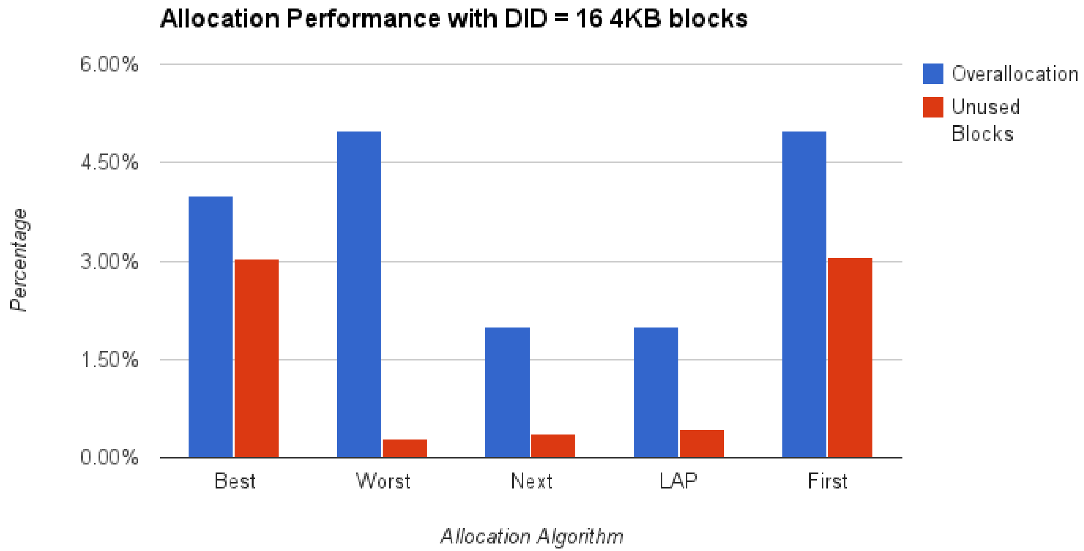

Figure 5.8: Percentage of unusable free blocks over total blocks in the disk

	Best Fit	First Fit	Next Fit	LAP Fit	Worst Fit
Total Disk blocks	272642	275354	267399	267399	275354
Total Free blocks	10498	13210	5255	5255	13210
% of free block over total blocks	4%	5%	2%	2%	5%
unusable blocks	8287	8437	936	1166	819
% of unusable blocks	3.04%	3.06 %	0.35%	0.44%	0.30%
Total usable blocks	2211	12391	4319	4089	4779
% of usable blocks over total free blocks	21.06%	36.13 %	82.19 %	77.81%	93.80%

Table 5.1: Fragmentation with DID as 16 4KB blocks

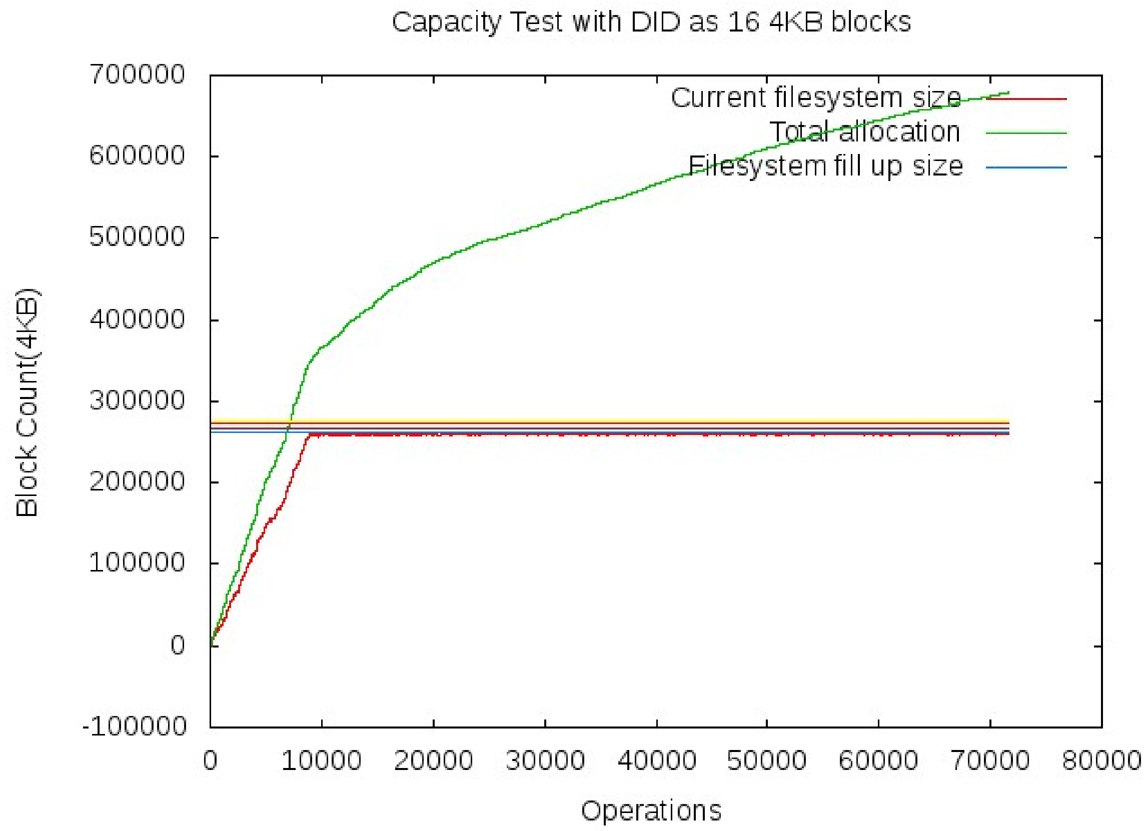

Figure 5.9: Overallocation & cleaning performance for DID as 16 4KB blocks

Brown line shows the over allocation for Best Fit, yellow line for Worst Fit and First Fit, while the dark blue line for Next Fit and LAP Fit

and coalesces the blocks to create bigger extents which will be used for the subsequent allocations. Also the cleaning algorithm implicitly cleans blocks with total capacity 1.5 GB !

Fragmentation produced by all the algorithms show different characteristics. Last Accessed Position Fit and Next Fit and Worst Fit algorithm mark the least number of blocks as unusable while the best fit and first fit algorithm results in most block of the disk to be marked as unusable this is shown in Figure 5.8. Most of the extents for these algorithms have less than 16 blocks, and since the value of DID is 16, any extent with less than 16 blocks will be marked as unusable and hence will not be

	Best Fit	First Fit	Next Fit	LAP Fit	Worst Fit
Total Disk blocks	327693	327693	288371	288371	340800
Total Free blocks	65549	65549	26227	26227	78656
% of free block over total blocks	25%	25%	10%	10%	30%
unusable blocks	61120	65891	8651	6708	62005
% of unusable blocks	18.65%	19.33%	3.00%	2.33%	18.92%
Total usable blocks	4429	12765	17576	19519	3544
% of usable blocks over total free blocks	6.76%	16.23%	67.01%	74.42%	5.41%

Table 5.2: Fragmentation with DID as 512 4KB blocks

used in further allocation decision unless coalescing happens on adjacent blocks to create bigger extents.

5.3.2 Test with DID as 512 4KB blocks

Apart from the DID value of 16 blocks of 4KB each, the other value of DID which we chose is 512 blocks of 4KB. This value was calculated depending on the Drive Transfer Rate (138MB/s) and the rotational speed (7200 rpm) for the current drives[10], from these parameters, the amount of data fetched from every rotation is close to 1MB and we chose this value to calculated the value of DID.

For this new value of DID it can be observed from Figure 5.10 and Table 5.2 that different values of overallocation were needed to run a 1GB workload on this filesystem. The lowest value of overallocation(10%) was observed for Next fit and Last Accessed Position fit algorithm while the highest overallocation(30%) was needed for Worst fit allocation algorithm. As can be seen from Figure 5.11 the total amount of allocation requests are almost 2.5 times the fill-up size of the filesystem. The implicit cleaning frees up blocks when the files are deleted. Cleaning would also coalesce data blocks to create bigger extents which would be used later for coalescing.

For this value of DID also, Last Accessed Position Fit and Next Fit algorithm

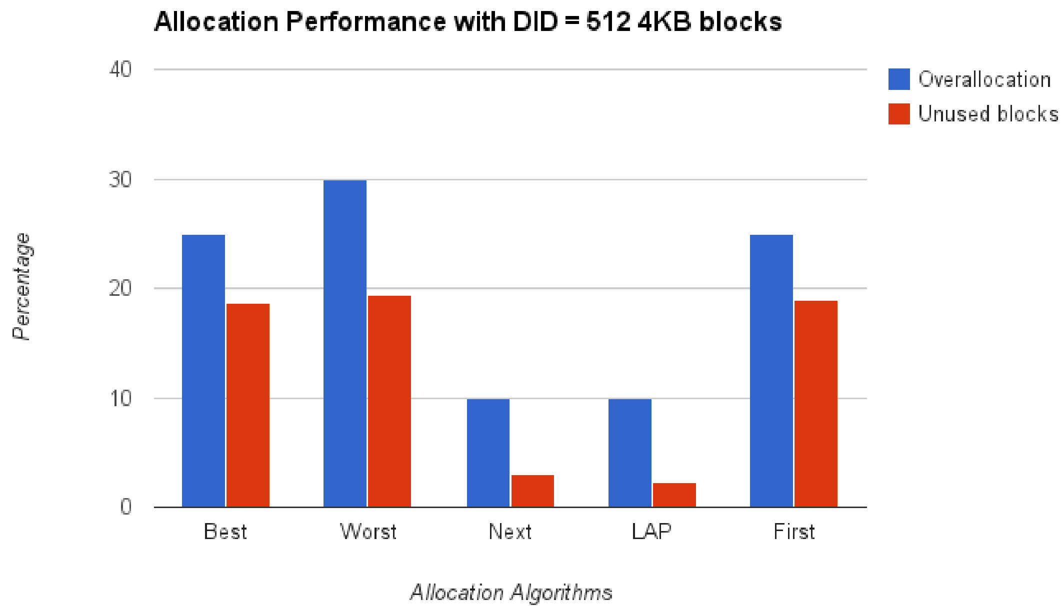

Figure 5.10: Percentage of unusable free blocks over total blocks in the disk

mark the least number of blocks as unusable while the best fit, first fit and next fit algorithm results in most block of the disk to be marked as unusable this is shown in Figure 5.10. Most of the extents for these algorithms have less than 512 blocks, and since the value of DID is 512, any extent with less than 512 blocks will be marked as unusable and hence will not be used in further allocation decision unless coalescing happens on adjacent blocks to create bigger extents.

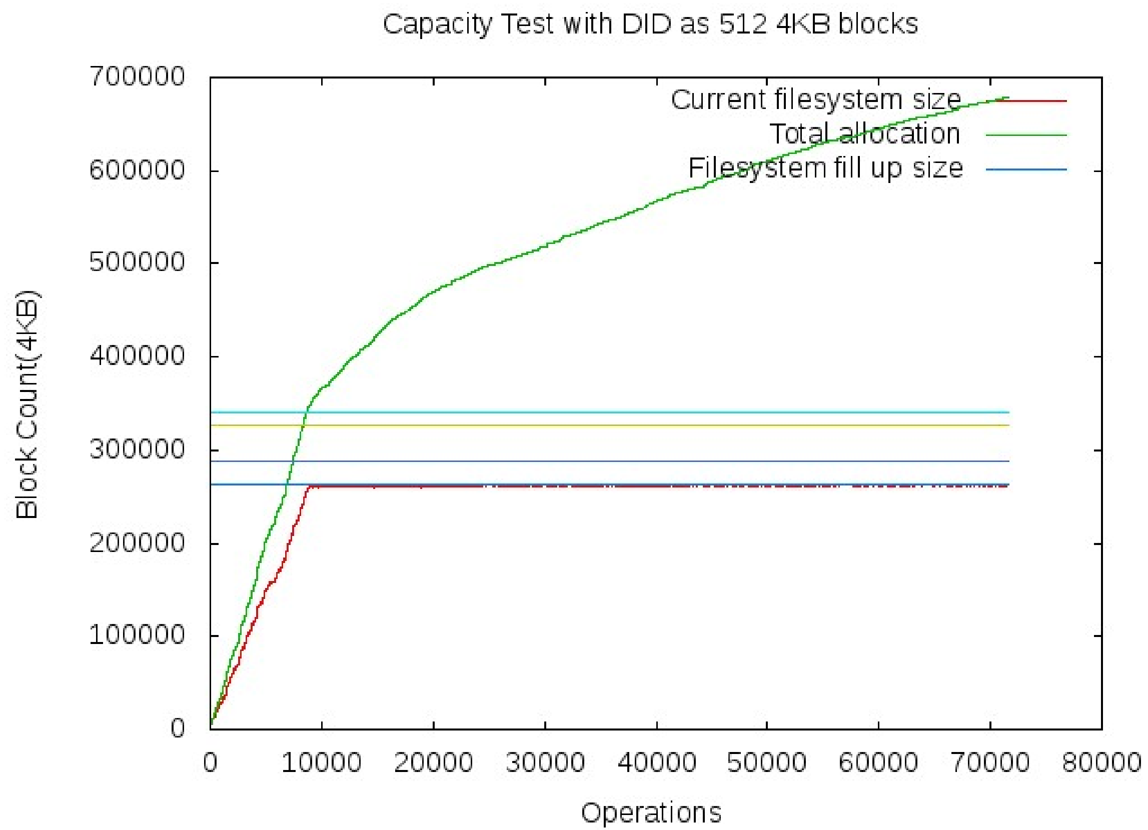

Figure 5.11: Overallocation & cleaning performance for DID as 512 4KB blocks

Yellow line shows the over allocation for Best Fit and First Fit Algorithm, light blue line for Worst Fit, while the dark blue line for Next Fit and LAP Fit

6

Conclusion

From the results and analysis, it can be concluded that a filesystem can be designed and developed over this extreme interface, Caveat Scriptor. Although this interface might corrupt data on downstream sector, but if the allocation decisions are taken correctly, then overwrites can be avoided and existing live data on the disk would still be consistent.

Filesystems can easily be implemented on disk supporting Caveat Scriptor model by doing an over-allocation for every allocation request. This over allocation will mitigate the shingling effects of the writes and then these blocks, which are serve as padding for the actual user data can later be freed when the write request finishes.

Caveat Scriptor also provides a simple interface when compared to Strict Append model, Strict Append model will require extensive changes in the disk interface and disk firmware. Firmware changes will be required to maintain the position of the current write pointer inside the band. Interface changes will be required to fetch and set the position of the write pointer apart from fetching the disk drive parameters. Caveat scriptor can be used with an interface similar to the one used in the current disk drive technology, however minor modification will be needed to

return the important disk parameters as DID and DNOR so that a filesystem can be implemented over these disk.

It can also be concluded that the cleaning cost associated with the Caveat Disk is minimal when compared to the strict append model. In the strict append model, data is always written on the drive in a log structured nature and hence cleaning is always necessary to perform garbage collection and free dead data blocks. Caveat Scriptor provides a technique where garbage collection happens implicitly and hence provides simpler and efficient cleaning semantics.

Shingling nature of the disk will definitely cause fragmentation in the file system and some blocks will be marked as unusable because the number of contiguous free blocks is less than DID. Using the correct block allocation algorithm will help in minimizing this unused capacity. This unused capacity of the caveat drive can also be mitigated by over-allocating drive capacity by a small percentage so that workloads meant for actual capacities can successfully be run on a Caveat Scriptor drive.

7

Future Work

Current implementation for Caveat Scriptor is based on a FUSE[5] based file system, one of the first enhancement will be into to tweak ext4[4, 3] filesystem to support caveat scriptor. This can be done by changing the block allocation technique in ext4. Interfaces defined in ext4_maps_blocks() should be explored to identify the required changes.

Another modification will be to enhance buffercache for Caveat Scriptor SMRfs to support segments of files rather than to support full files in the buffercache. This might lead to interesting results in the allocation and deallocation because a file now can reside in multiple contiguous portions of the disk unlike what's happening with the current implementation. This might exhibit interesting fragmentation results to be observed in the filesystem. Efficient use of buffercache and pre allocation of blocks should help in reducing fragmentation for large files.

Other filesystems as xfs[11] and btrfs[2] can also be modified to add support for caveat scriptor.

Bibliography

[1] "Block bitmap." [Online]. Available: http://www.nongnu.org/ext2-doc/ext2.html#BLOCK-BITMAP." [Accessed 1-September-2013].

[2] "Btrfs." [Online]. Available: https://btrfs.wiki.kernel.org/index.php/Main_Page." [Accessed 2-January-2014].

[3] "ext4." [Online]. Available: http://en.wikipedia.org/wiki/Ext4." [Accessed 31-September-2013].

[4] "Ext4 (and ext2/ext3) wiki." [Online]. Available: https://ext4.wiki.kernel.org/index.php/Main_Page." [Accessed 31-September-2013].

[5] "Filesystem in userspace." [Online]. Available: http://fuse.sourceforge.net/." [Accessed 1-September-2013].

[6] "fs_mark." [Online]. Available: http://fsmark.sourceforge.net/." [Accessed 1-January-2014].

[7] "Nilfs." [Online]. Available: http://nilfs.sourceforge.net/en/index.html." [Accessed 10-September-2013].

[8] "Probe: Parallel reconfigurable observarional environment." [Online]. Available: http://marmot.pdl.cmu.edu/." [Accessed 1-January-2014].

[9] "vdbench." [Online]. Available: http://vdbench.sourceforge.net/." [Accessed 1-January-2014].

[10] "Wd caviar black spec sheet." [Online]. Available: http://www.wdc.com/wdproducts/library/SpecSheet/ENG/2879-701276.pdf." [Accessed accessed 2-January-2014].

[11] "Xfs." [Online]. Available: http://xfs.org/index.php/Main_Page." [Accessed accessed 2-January-2014].

[12] A. Amer, J. Holliday, D. D. E. Long, E. L. Miller, J.-F. Pris, and T. Schwarz, "Data management and layout for shingled magnetic recording," *IEEE Transactions on Magnetics*, vol. 47, no. 10, oct 2011.

[13] E. A. Dobisz, Z. Bandic, T.-W. Wu, and T. Albrecht, "Patterned media: Nanofabrication challenges of future disk drives," *Proceedings of the IEEE*, vol. 96, no. 11, pp. 1836–1846, Nov 2008.

[14] T. Feldman and G. Gibson, "Shingled magnetic recording: Areal density increase requires new data management," *;login: the USENIX Association newsletter*, vol. 38, no. 3, pp. 22–30, June 2013.

[15] G. Gibson and G. Ganger, "Principles of operation for shingled disk devices," Carnegie Mellon University Parallel Data Lab, Tech. Rep. CMU-PDL-11-107, April 2011.

[16] G. Gibson, G. Grider, A. Jacobson, and W. Lloyd, "Probe: A thousand-node experimental cluster for computer systems research," vol. 38, no. 3, June 2013.

[17] G. Gibson and M. Polte, "Directions for shingled-write and two-dimensional magnetic recording system architectures: Synergies with solid-state disks," Carnegie Mellon University Parallel Data Lab, Tech. Rep. CMU-PDL-09-104, May 2009.

[18] J. Katcher, "Postmark: A new file system benchmark," 1997, technical report TR3022. Network Applicance Inc. October 1997.

[19] Q. M. Le, K. SathyanarayanaRaju, A. Amer, and J. Holliday, "Workload impact on shingled write disks: All-writes can be alright," in *Proceedings of the 2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, ser. MASCOTS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 444–446. [Online]. Available: http://dx.doi.org/10.1109/MASCOTS.2011.58. [Accessed 31-September-2013].

[20] M. K. McKusick, K. Bostic, M. J. Karels, and J. S. Quarterman, *The Design and Implementation of the 4.4BSD Operating System*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1996.

[21] M. Rosenblum and J. K. Ousterhout, "The design and implementation of a log-structured file system," *ACM Trans. Comput. Syst.*, vol. 10, no. 1, pp. 26–52, feb 1992. [Online]. Available: http://doi.acm.org/10.1145/146941.146943. [Accessed 1-September-2013].

[22] R. Rottmayer, S. Batra, D. Buechel, W. Challener, J. Hohlfeld, Y. Kubota, L. Li, B. Lu, C. Mihalcea, K. Mountfield, K. Pelhos, C. Peng, T. Rausch, M. A. Seigler, D. Weller, and X. Yang, "Heat-assisted magnetic recording," *Magnetics, IEEE Transactions on*, vol. 42, no. 10, pp. 2417–2421, Oct 2006.

[23] A. Suresh, G. Gibson, and G. Ganger, "Shingled magnetic recording for big data applications," Carnegie Mellon University Parallel Data Lab, Tech. Rep. CMU-PDL-12-105, May 2012.

[24] D. A. Thompson and J. S. Best, "The future of magnetic data storage technology," *IBM J. Res. Dev.*, vol. 44, no. 3, pp. 311–322, may 2000. [Online]. Available: http://dx.doi.org/10.1147/rd.443.0311. [Accessed 14-January-2014].

[25] C. Walter, "Kryder's law," July 2005. [Online]. Available: http://www.scientificamerican.com/article/kryders-law/. [Accessed 10-February-2014].

[26] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proceedings of the USENIX Symposium on Operating System Design and Implementation (OSDI).* USENIX, dec 2002.

[27] S. wiki, "Aging tool." [Online]. Available: https://wiki.pdl.cmu.edu/SMR/SMRfsTools." [Accessed 31-September-2013].