Hybrid system analysis and design for scale-out storage environments

Niu, Junpeng

2018

Niu, J. (2018). Hybrid system analysis and design for scale-out storage environments.
Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/80426>

<https://doi.org/10.32657/10220/46495>

HYBRID SYSTEM ANALYSIS AND DESIGN FOR SCALE-OUT STORAGE ENVIRONMENTS

JUNPENG NIU

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University

in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

2018

*To my family,
for their encouragement and love.*

First of all, I would like to express my great appreciation to my supervisor, Prof. Lihua Xie, for his continuous guidance, encouragement and support throughout my research efforts towards this dissertation. Without his inspiration, deep insight, and invaluable advice, this thesis would not have been possible. I have benefited tremendously from his extensive knowledge, professional experience, and strong commitment towards the excellence of research.

Special thanks are given to my co-advisor, Dr. Jun Xu from Western Digital. I am flabbergasted to his knowledge, open mind, kindness, and professionalism. He has been always helpful for my study, work and life. It has always been meaningful and fruitful to discuss and work together with him. I really wish to continue to learn from him if possible.

I would also like to express my appreciation to my managers, Dr. Jie Yu, Dr. Guoxiao Guo and Dr. Jianyi Wang for their advice, encouragement and help. I am thankful to my friends, Dr. Hui Li and Dr. Shu Feng to introduce me such a great program and amazing company. My thanks also go to my colleagues Ms. Wai Ee Wong, Dr. Jie Wan, Mr. Yanfeng Zhang, Mr. Shanjiao Jiang, Dr. Kun Ma, Dr. Bin Hu, and Mr Chih Wei Ling, for their help and knowledge sharing.

Moreover, I would like to take this opportunity to record my sincere thanks to my school, School of Electrical and Electronic Engineering, Nanyang Technological University, my company, Western Digital Corporation, and Economic Development Board (EDB), Singapore for providing me this amazing opportunity and financial assistance in pursuit of my doctoral degree.

Last but not least, I extend my utmost gratitude to my parents, Mr. Wenliang Niu, and Ms. Runtao Liu for their upbringing, love and support. I have to thank my parents in law, Mr. Huisheng Guo and Ms. Jianping Huang for their support and effort, especially for their help in taking care of my children during my study. I must give my thanks to my wife, Dr. Tiantian Guo and my two sons, Yunhan Niu and

Yunlei Guo for their unconditional love, understanding and supports. I would also like to say thanks to my two younger brothers, Mr. Jingpeng Niu and Mr. Qipeng Niu for their help to take care of my parents in hometown during my busy time. They are the most important people in my world.

Table of Contents

Acknowledgements	i
Table of Contents	iv
Summary	xi
List of Figures	xiv
List of Tables	xviii
1 Introduction	1
1.1 Background	1
1.2 Motivation of Dissertation	5
1.3 Contribution	7
1.4 Thesis Organization	9
Nomenclature	1
2 Literature Survey	11

2.1	SMR Drive	11
2.2	Hybrid Storage Architecture	14
2.2.1	SSD Caching Method	17
2.2.1.1	SSD as a Read-only Cache	18
2.2.1.2	SSD as a Write-back Cache	19
2.2.2	SSD Tiering Method	24
2.2.3	SSD Hybrid Method	27
2.2.4	Other Structures with Special Purposes	29
2.2.5	Hybrid Structures Using Various NVMs	31
2.3	Algorithms Utilized in Hybrid Storage Systems	32
2.3.1	Data Allocation	32
2.3.2	Address Mapping Algorithms	33
2.3.3	Hot Data Identification and Data Migration Process	35
2.3.3.1	Access Frequency	37
2.3.3.2	Access Frequency and Recency	37
2.3.3.3	Frequency/Recency with Other Factors	37
2.3.3.4	Data Migration Algorithms	38
2.3.4	Cache Scheduling Algorithms	39
2.3.4.1	Cache Scheduling Algorithms in HDD	39
2.3.4.2	Cache Scheduling Algorithms in SSD	40

2.3.4.3	Cache Scheduling Algorithms in Hybrid Storage Sys-	
	tems	40
3	Theoretical analysis of SMR drive performance	42
3.1	Introduction	42
3.2	Background	45
3.3	Problem Formulation	47
3.3.1	Workload Property Formulation	48
3.3.2	Disk Processing Time Formulation	50
3.3.3	Disk Resource Constraints	51
3.3.3.1	Internal Queue Length Constraint	51
3.3.3.2	Disk Capacity Usage Constraint	52
3.3.4	Performance Evaluation	53
3.4	Analysis	54
3.4.1	Disk Processing Time Analysis	54
3.4.1.1	Seek Time Distribution	55
3.4.1.2	Rotational Latency Distribution	57
3.4.1.3	Disk Processing Time under Different Queue Length	58
3.4.2	P_b Calculation and M/G/1/K Queue Analysis	61
3.4.3	Performance Modeling under Different User Batch Write Sizes	67
3.4.4	Performance Modeling under Different GC Batch Write Sizes	70
3.4.5	Performance Modeling under Different Disk Fullness Statuses	72

3.5	Simulation Setup	72
3.6	Results and Discussion	73
3.6.1	Performance Analysis under Different User Write Batch Sizes .	74
3.6.2	Performance Analysis under Different GC Batch Write Sizes .	78
3.6.3	Performance Analysis under Different Disk Fullness Statuses .	80
3.7	Conclusion	82
4	Optimal Selection of GC Dirty Threshold to Balance Energy Consumption and Space Release	84
4.1	Introduction	84
4.2	Background	86
4.3	Problem Formulation	87
4.3.1	Background GC Process	89
4.3.2	Foreground GC Process	90
4.4	Results and Analysis	91
4.4.1	Results under Different Workload Properties	92
4.4.1.1	Request Size	92
4.4.1.2	Read Ratio	93
4.4.2	Results under Different System Settings	93
4.4.2.1	Disk Status	95
4.4.2.2	GC Batch Size	95
4.4.2.3	Idle Time Threshold	96

4.4.2.4	Host Write Batch Size	99
4.5	Conclusion	99
5	Hybrid Storage System Simulation and Comparison	101
5.1	Background	101
5.2	Simulation Design and Setup	103
5.3	System Specification	104
5.4	Performance Comparisons under Different Caching Policies	106
5.4.1	Performance	108
5.4.2	SSD Lifespan	110
5.4.3	Energy Consumption	112
5.5	Performance Comparisons under Different System Architectures	113
5.5.1	Performance	114
5.5.2	SSD Lifespan	116
5.5.3	Energy Consumption	117
5.6	Conclusion	120
6	New Hybrid Disk Array Structures with Consistent Performance and Prolonged Life	121
6.1	Introduction	121
6.2	Background	124
6.3	System Architecture Design	127

6.3.1	Performance Stripe Formulation	128
6.3.2	Disk Array Construction	130
6.3.3	Data Allocation and Migration	131
6.3.4	EC/RAID Scheme Consideration	133
6.4	Model Analysis	134
6.4.1	System Settings	135
6.4.2	Analytical Modeling	136
6.5	Simulation Results	141
6.5.1	The Impact of Sequential Ratio and Write Ratio of the Workload	141
6.5.2	The Impact with Threshold Settings for Random and Sequential Properties	143
6.5.3	The Impact of Hot Read/Write Ratio	145
6.6	System Analysis and Discussion	145
6.7	Conclusion	150
7	Hybrid Storage System Performance Improvement through Online Fuzzy Logic Control and Decision Tree	151
7.1	Introduction	151
7.2	Background	154
7.2.1	Hot data identification and data migration algorithms	154
7.2.2	Block level workload characterization	155
7.2.3	Performance evaluation metrics	155

7.2.4	Control algorithms for system performance improvement . . .	156
7.3	Methodology	156
7.3.1	Gathering of System Performance for Different Kinds of Work- loads	157
7.3.2	Evaluation of Different Kinds of Caching Algorithms	158
7.3.3	Selection of Block Level Workload Characterization	158
7.3.4	Process of Training Workload Properties	161
7.3.5	Online Fuzzy Control Caching Algorithm	162
7.3.5.1	Workload Properties Calculation	162
7.3.5.2	Fuzzification of Inputs	163
7.3.5.3	Fuzzy Rules Generation and Usage	163
7.3.5.4	Defuzzification of Outputs	166
7.4	Evaluation	167
7.4.1	Evaluation under Different Workload Properties	167
7.4.2	Evaluation under Different IOPS	169
7.5	Conclusion	171
8	Conclusion and Future Directions	172
8.1	Conclusion	172
8.2	Future Directions	174
	Author's Publications	178

Data center storage architectures face rapidly increasing data volume and Quality of Service (QoS) expectation today. Hybrid storage systems have turned out to be one of the most democratic choices in fulfilling these requirements. A mixture of various types of storage devices and structures enables architects to address performance and capacity concerns of users within one storage infrastructure. There are lots of architectures, such as tiered storage and caching storage, etc. and policies, such as caching policy and replacement policy, etc. in a hybrid storage system. It is difficult to employ a general mathematical model to capture all the features of the storage architectures. Therefore, simulation is necessary to analyze the system performance and guide the system design.

In a hybrid storage system, in addition to the performance requirement, the storage capacity provided is equally important. Shingled Magnetic Recording (SMR) is one of the techniques to enlarge the storage capacity with minimally increased cost. The batch process is applied to improve the SMR drive performance. The selection of different system parameters and policies affects the system performance and capacity efficiency. Thus, an analytical model is developed based on queuing model to study the SMR drive performance. Simulation is further conducted to validate the model and check the performance impacts of various drive parameters.

In an SMR drive, due to its append-only write property, the data update is routinely handled through a log-structured manner, which makes the original data become invalid and dirty. Various types of Garbage Collection (GC) methods are therefore introduced to clean the dirty data and thus release the disk space. However, the necessity of GC process performed remains an issue, as the process may increase the energy consumption and downgrade the performance. Thus, an analytical model and a simulator are built to study the decision of the GC process under various kinds of workload environments and system settings for overall power consumption reduction and disk space saving.

Based on the performance analysis of individual devices, the system level performance of the hybrid storage system can be studied. A flexible hybrid storage system simulator is designed and developed to simulate various kinds of hybrid storage architectures, including Solid State Drive (SSD) tiering method, SSD caching method and SSD hybrid method, various caching policies, such as read-only and write-back, and various hot data identification and data migration policies. The performances of these architectures and algorithms are evaluated and compared under different types of workload environments. The comparison results can be utilized as the benchmark for analyzing other types of hybrid storage systems, which are the extension of these basic storage architectures.

Note that the conventional hybrid storage systems do not fully utilize the sequential access properties and the non-limited write cycles of Hard Disk Drive (HDD). An innovative approach is proposed to configure HDDs and SSDs in a hybrid structure such that the advantages of both sides can be fully utilized, i.e., the fast IO access of SSD (in particular for random access) and the non-limitation of write cycles of HDD. By carefully designing the disk data stripes, the (sequential) performance requirement for HDDs and SSDs can be matched to a certain degree. Therefore, they can be placed in the same array/pool without considering the high/low tiers or fast/slow cache.

In a hybrid storage system, the caching policies, such as hot/cold identification and data migration, play a critical role in system performance. In particular, the migration size is one of the key factors. Fixed size of data migration typically cannot provide good performance when the workload properties change significantly and frequently. We design a hybrid caching algorithm based on the fuzzy control and decision tree which can adaptively adjust the data migration policies according to the workload properties. The fuzzy rules can be automatically generated through the training results of the decision tree classification and regression algorithms.

List of Figures

1.1	General algorithms for hybrid storage systems	5
2.1	The data layout of SMR drive	13
2.2	The overall categories of the hybrid storage architectures	15
2.3	The difference between the host and device controlled policies	17
2.4	The data flows in the system with SSD as a cache to HDD	18
2.5	The data flows in the system with SSD as a read and write split cache to HDD	22
2.6	The data flows in the system with SSD as a tier	26
2.7	The data flows in the system with SSD hybrid methods	29
2.8	The data flows in the system with HDD as a write cache of SSD . . .	30
3.1	The illustrative diagram for request queuing in SMR drive with GC process	47
3.2	The method to select different GC policies	52
3.3	The CDF and PDF of disk positioning times under different queue lengths.	61

3.4	The disk positioning times and number of requests in disk queue under different IOPS.	62
3.5	The overall structure of the SMR drive simulator	73
3.6	SMR drive performance under different workload properties and write batch sizes.	75
3.7	SMR drive performance under different workload properties and GC batch write sizes.	77
3.8	SMR drive performance under different workload properties and disk statuses.	79
4.1	A feed-forward controller to control the GC dirty ratio threshold . . .	87
4.2	Cost savings under different request sizes and r_{dirty}^t	92
4.3	Cost savings under different read ratios and r_{dirty}^t	94
4.4	Cost savings under different read ratios and r_{dirty}^t with cost ratio 50. .	94
4.5	Cost savings under different disk statuses and r_{dirty}^t	95
4.6	Cost savings under different disk statuses and r_{dirty}^t with cost ratio 50.	96
4.7	Cost savings under different GC batch sizes and r_{dirty}^t	96
4.8	Cost savings under different batch sizes and r_{dirty}^t with cost ratio 50. .	97
4.9	Cost savings under different idle time thresholds and r_{dirty}^t	97
4.10	Cost savings under different idle time thresholds and r_{dirty}^t with cost ratio 50.	97
4.11	Cost savings under different host write batch sizes and r_{dirty}^t	98

4.12	Cost savings under different host write batch sizes and r_{dirty}^t with cost ratio 50.	98
5.1	The overall structure of the hybrid storage system simulator	103
5.2	System performance under different caching policies for SPC1C traces. 107	
5.3	SSD lifespan under different caching policies for SPC1C traces.	110
5.4	System energy consumption under different caching policies for SPC1C traces.	112
5.5	System performances under different storage architectures for SPC1C traces.	115
5.6	The system performances under different hot data identification policies for SPC1C traces.	116
5.7	SSD lifespan usage under different storage architectures for SPC1C traces.	117
5.8	The SSD lifespan usage under different hot data identification policies for SPC1C traces.	118
5.9	System energy consumption under different storage architectures for SPC1C traces.	118
6.1	IO directions between SSD and HDD	128
6.2	An example of performance stripes	129
6.3	An example of hybrid disk array structure	131
6.4	A general framework of the data allocation	133

6.5	Data allocation for separated random and sequential accesses	134
6.6	Efficiency and lifespan savings under different sequential and read ratios.	142
6.7	The performances and lifespan prolonged under different read and write ratios to HDD	144
6.8	Performance efficiency and lifespan prolonged under different hot read and write ratio under 3:1 HDDs/SSD ratio.	146
6.9	Performance efficiency under different hot read and write ratios under 2:1 HDDs/SSD ratio.	147
7.1	The fuzzy control block diagram for the caching system	157
7.2	The overall workflow of the fuzzy logic controlled caching system . . .	158
7.3	Training results with decision tree classification	161
7.4	Fuzzication of the inputs and outputs of the control system	165
7.5	Online fuzzy control caching system overview	165
7.6	Performances under different workload combinations and caching algorithms	168
7.7	Performances under different workload combinations and caching algorithms	169
7.8	Performances under different IOPS and caching algorithms	170

List of Tables

1.1	The performance, power consumption, price and endurance of various types of non-volatile memories and HDD	3
2.1	The literature of hybrid storage systems (HDDs + SSDs)	15
2.2	Comparison of the hybrid storage systems with SSD caching method	16
2.3	Comparison of the hybrid storage systems with SSD tiering method .	25
2.4	Two aspects that need to be considered during designing of hot data identification algorithm	33
2.5	Other workload properties that need to be considered during storage system design	34
2.6	Comparison of hot data identification algorithms	36
3.1	SMR drive and input workload related parameters	49
3.2	SMR drive setting parameters	50
4.1	The parameters utilized in the problem formulation and simulation .	88
5.1	The hardware specification of HDD in the hybrid storage system . . .	105

5.2	The hardware specification of SSD in the hybrid storage system . . .	105
5.3	The different types of workloads and their properties	105
5.4	Performance comparison under different caching policies for financial trace	107
5.5	Performance comparison under different caching policies for web search- ing trace	107
5.6	Performance comparison under different system architectures for fi- nancial trace	114
5.7	Performance comparison under different system architectures for web searching trace	114
6.1	An example of SSD specification	135
6.2	An example of HDD specification	135
6.3	Notations of incoming workload properties	136
6.4	The formulas to calculate the percentages of read and write requests to different devices	136
6.5	The percentages of random read/write requests to different devices .	137
6.6	The percentages of sequential read/write requests to different devices	137
6.7	The percentage of read and write requests to different devices	138
6.8	The system efficiency and SSD lifespan saving	140
7.1	The performance metric and description	159
7.2	The workload properties and description	160

7.3	The generated fuzzy rules	166
7.4	Workload properties generation	167

Chapter 1

Introduction

1.1 Background

With the development of Internet of Thing (*IoT*) technologies and big data analysis, the amount of data generated by IoT devices increases dramatically. One of the reports from IDC [1] sees the digital universe doubling in size every two years, and until 2020, the data may increase to 40 zettabytes. Such tremendous and assorted data requirements create challenging issues for the design of storage infrastructures. In addition to the requirement of storage capacity, data accessing speed and reliability are also key factors influencing the user experience.

Due to the mechanical limitations of Hard disk driver (*HDD*), the traditional storage architecture turns out to be the major performance bottleneck in the data intensive systems. With the fast evolution of Flash memory technologies [2], *SSD* is going to replace HDD in several data centers (*DCs*). This is on the fact of that the IOPS of SSD can be hundreds or even thousands times faster than that of HDD, and throughput can be times over HDD. Additionally, SSD reduces the energy

consumption and provides stable execution under vibration. However, due to the high cost per GB, limited write cycles and reliability issues of SSD [3], HDD still plays an important role in modern storage systems.

In order to provide both large storage capacity and fast access speed with the minimal cost, it is vital to utilize the high capacity of low tier devices with lower accessing speed and lower cost per GB, e.g. HDD, and the high performance of high tier devices with lower capacity and higher cost per GB, e.g. SSD. Hybrid storage architecture is one of the solutions that can provide comparable or close system performance with all fast tier devices while giving nearly same storage capacity with all lower tier devices. It usually has the ability to promote or demote different types of data across different tiers of storage devices. The data movement is managed via the host or the drive itself according to the performance and capacity requirements. The hybrid storage system is the main-stream in modern data centers and will continue to play an important role in the storage infrastructure if the performances and cost differences of storage devices still exist.

In early days, high-end HDDs (such as 15k RPM and 10k RPM) acted as the performance tier, and low-end HDDs (such as 7200 Rate per Minute (*RPM*) and 5400 RPM) acted as the capacity tier [4]. Nowadays, NAND Flash [2] SSDs are employed to replace high-end HDDs. The next generation of hybrid storage systems will incorporate with the emerging Non-volatile Memory (*NVM*) technologies, such as Phase Change Memory (*PCM*) [5], Spin-Transfer Torque Magnetic random-access memory (*STTMRAM*) [6], Resistive RAM (*RRAM*) [7], etc. Table 1.1 lists the performance and price of some emerged NVMs and HDD. As the performance cost ratio, which is the ratio of performance, the combination of random access speed (*IOPS*) and sequential access speed (throughput) for various system requirements over the price per GB, of NVMs is increasing, these NVMs with fast accessing speed can be used as the performance tier [8,9] or cache [10–13] in modern hybrid storage

Table 1.1: The performance, power consumption, price and endurance of various types of non-volatile memories and HDD

	STTMRAM	PCMS 3D Xpoint	RRAM	Flash NAND	HDD
Read latency	10 – 20 ns	50 – 100 ns	250 ns	10 ⁵ ns	3 ms
Power consumption	Medium	Medium	Medium	Very high	Highest
Price (2016)	200 – 3000/ Gb	$\leq$ 0.5/ Gb	100/ Gb	$\leq$ 0.05/ Gb	$\leq$ 0.002/ Gb
Endurance (Nb cycles)	10 ¹²	10 ⁸	10 ⁶	10 ⁵	N.A

systems.

Today, although there are hybrid drive products that consist of HDD and NAND Flash, DCs still prefer to design their own hybrid structures in the system level. In the system, SSD is still the first choice as the performance tier, HDD and the high capacity SMR drives are often utilized as the backup tier [14–17]. Many industry players [18–25] integrate the SSD devices into the traditional storage systems as the faster tier or cache to improve the system performance with slightly more cost. Meanwhile, lots of researchers, as listed in Table 2.1, have also proposed various structures and algorithms to improve one or few aspects of hybrid storage systems. In this thesis, we concentrate on the hybrid storage structures that combine HDD/SMRs and SSDs, despite the fact that the general idea can be applied to a wide range of hybrid storage systems.

A general hybrid storage architecture is shown in Fig. 1.1, where different algorithms and policies are used to manage the external and internal data for the tier and caching storage architectures. There are mainly two types of hybrid storage architectures, tiered storage architecture (tiering method) and cache storage architecture (caching method). The fundamental distinction between them are: a). Tiering method moves data to the fast storage area as opposed to copying the data

in caching method. b). The time duration that the hot data, which is the frequently and/or recently accessed data, stay in the faster tier device is normally longer with a tiering method compared with a caching method. However, their performances are chiefly influenced by the following four aspects:

- Firstly, data allocation policies are essential to control the data flow among various devices. The data allocation in a tiering method allocates the data based on the hotness of the data. Meanwhile, the data allocation in a caching method distributes the data to different devices in light of the caching policies utilized, such as read-only and write-back.
- Secondly, address translation mechanism is needed in a hybrid storage system to keep the different locations in different devices for the same data. The design of address translation is important to the speed of data retrieval and the size of memory utilization to store the translation information.
- Thirdly, due to the size limitation of SSD cache/tier compared with main storage HDDs, the SSD cache/tier is better to only store the hot data. Therefore, it is important to have a suitable hot data identification method to identify whether the coming data is hot or not. When the hot data is detected, the data needs to be promoted when necessary. A data migration policy is needed to control the hot data flow to SSD to improve the future access efficiency. Meanwhile, the cold data, which is less frequently and recently accessed data, will be migrated to HDD to save the SSD space. The hot data identification and data migration consider the trade-off between the cost of data migration and the benefit of accessing speed of the faster device, which affect the overall system performance.
- Lastly, a cache scheduling algorithm is necessary for controlling the queuing behaviors, such as the queue size, synchronization, and execution sequence.

Figure 1.1: General algorithms for hybrid storage systems

The designed scheduling algorithm affects the usage efficiency of the storage devices and the cache inside.

1.2 Motivation of Dissertation

As aforementioned in the background, the hybrid storage system is the main-stream storage architecture in the DC design. Industry players and researchers have contributed a tremendous number of structures and algorithms to improve the system performance, including the response time, system reliability, and energy consumption. With the huge growth of storage-related demands for capacity, performance, functionality and flexibility, especially the demand of big data analytic, mobility, social platform integration, and IoT related data growth, the requirements for the storage system will be more and more challenging and the complexity of the system design will keep increasing. It is difficult to design a storage architecture to meet all of the client requirements. Meanwhile, a lot of efforts have to be done to build up the storage system and adjust the system parameters to provide user satisfied performance with reasonable cost.

In lieu of the difficulty of designing and comparing modern storage systems, analytical model and simulation become important for the system design and parameter tuning. There are lots of storage system related simulators available nowadays, including single disk drive simulators, and complex data center storage simulators.

However, all these kinds of simulation tools only focus on one or few levels of the whole storage architecture. It is difficult to provide complete and detailed results for the performance analysis and comparison.

In a hybrid storage system, *SMR* drive plays an important role as the back-up tier, which has high-density and large capacity. It is important to know the performance, including the response time, IOPS, throughput and the energy consumption of the *SMR* drive to design the overall hybrid storage architecture and algorithms. In an *SMR* drive, there are different kinds of algorithms, such as garbage collection (*GC*), defragmentation (*DF*), and batch process, and different types of parameters, such as *GC* threshold, user write batch size, *GC* batch write size and idle time threshold. It is not easy to design and compare different kinds of *SMR* drives through actual implementation. Simulation is necessary for the *SMR* drive design as it can show and compare the impact of different kinds of algorithms and settings on the drive performance. However, tuning parameters through simulation is also time consuming, theoretical analysis becomes important as a guide to the drive design and parameter settings.

In a hybrid storage system consisting of *HDDs* and *SSDs*, it is better to allocate the hot data to *SSD* to improve the chance of future access to *SSD* and improve the overall system performance. However, the writing data might also have larger access frequency, which can be called hot write data. If we allocate these hot writing data to *SSD*, the lifespan of the *SSD* device will be consumed dramatically. Meanwhile, the processing speed of *HDD* for the sequential write data is not far behind compared with *SSD*. With the batch processing policy, we can group the random write requests into sequential one and improve the random write access speed of *HDD/SMR*. Thus, one possible design is to allocate the hot write data to *HDD*, and hot read data to *SSD*, which might prolong the lifespan of *SSD*, while maintaining the overall system performance.

Hot data identification and data migration are two important algorithms inside a hybrid storage system. The proper selection of migration size affects the overall system performance, as it needs to balance the migration benefit, such as the future cache/faster tier hit ratio, and the accessing speed, etc. and the migration cost, including reading data from HDD, writing data to SSD and the data transferring time. However, it is not easy to utilize one migration size to fit all kinds of workloads, as they may have different properties, such as larger migration size is suitable for the workloads with high re-access ratio, while is not suitable for workloads with a high random ratio. Thus, the design of an adaptive migration policy is an important step for the hybrid storage systems under varied workload environments.

1.3 Contribution

The main contributions of this thesis are as follows:

- A flexible storage system simulator is designed and developed. In the simulator, the conventional HDD and SMR drives are simulated. Different kinds of hybrid storage systems with various architectures and algorithms are simulated, and their performances are compared and analyzed.
- The SMR drive is formulated as an M/G/1/K queue with state dependent service time. The formulation can be utilized to analyze the performance of SMR drive under different workload environments and drive setting parameters. An SMR drive simulator is designed and developed to study the drive performance under different drive settings. The simulation results are utilized to validate the analytical model, and the validation shows the correctness of the model. Meanwhile, a new GC policy is proposed to solve the existing problem that the performance is degraded significantly when disk capacity is

nearly full. The SMR drive performance is studied and optimized under different drive settings, such as write batch size, GC batch size and disk status, etc., and under different workload environments, such as read ratio, request size and sequential ratio, etc., through theoretical and simulation approaches.

- An optimization problem is formulated to find the most suitable garbage ratio threshold for the GC process under different workload properties and drive parameters. The formulation is based on the cleaned disk space, energy consumption and the performance degradation caused by the GC process. The cost savings of the SMR drive with different settings are evaluated based on the analytical and simulation results, which can be utilized as a guideline for the SMR drive design.
- A new hybrid structure configuration method is proposed to arrange the HDDs and SSDs in a hybrid storage system. It aims to save the total cost of the system, while maintaining and improving the system performance and prolonging the SSD lifespan. A new data allocation method is introduced to allocate the data based on the data properties, such as hotness, read or write, and sequential or random. By fully utilizing the advantages of HDDs and SSDs, the SSD lifespan is prolonged and the overall system performance is maintained or even improved.
- A new data migration policy is proposed to improve the efficiency of data movements in hybrid storage systems. The migration size is determined through an online fuzzy control system, which allows the system to change its migration policies based on the incoming workloads. The fuzzy rules utilized inside the system are generated by analyzing the performance of the storage system under different kinds of workloads and cache policies. The fuzzy rules are generated automatically through a decision tree classification and regression model.

1.4 Thesis Organization

The rest of the thesis is organized as follows.

- In Chapter 2, the main hybrid storage architectures are reviewed, including the tiering method, caching method, hybrid method, and other special types of architectures. Algorithms and policies, such as hot data identification, data migration, data allocation, scheduling, and address mapping, etc. are introduced. Meanwhile, the devices utilized in the hybrid storage system, especially the SMR drive, are also reviewed.
- In Chapter 3, the simulation and theoretical analysis of the SMR drive are presented. The results from simulation and analytical model are presented and compared.
- In Chapter 4, an analytical model for energy consumption and the disk space cleaned by the GC process of SMR drive is presented. Optimal GC dirty ratio threshold is selected and the results under different kinds of GC thresholds are presented.
- In Chapter 5, the performances of different types of hybrid storage system architectures and algorithms are compared and analyzed including the overall response time, SSD lifespan and total energy consumption.
- In Chapter 6, a hybrid array that consists of HDDs and SSDs is presented. The performances, including IOPS, throughput, and the SSD lifespan utilization are studied and compared under different kinds of workload environments and system settings.
- In Chapter 7, the data migration size in hybrid storage systems is studied. An adaptive migration policy is proposed to make the hybrid storage system

fit with different kinds of workloads. The results of the proposed method are also compared with the fix size migration policies under varied workload environments.

- In Chapter 8, we summarize the thesis and point out the future research directions.

Chapter 2

Literature Survey

In this chapter, the previous research works related to the hybrid storage systems, particularly the ones consisting of HDDs and SSDs are reviewed. This chapter contains three sections. Firstly, the devices, especially the SMR drive, utilized inside the hybrid storage system are presented. Then, different types of hybrid storage architectures are introduced. Lastly, the algorithms and policies applied in the storage systems are described.

2.1 SMR Drive

To meet the rapid growing of data volume, SMR drive [26] [27] [28] is introduced to increase the storage capacity while minimizing the added storage cost. In the conventional HDD drive, the read header is substantially smaller than the write header, and the width of the read track is much narrower than the write track. The SMR drive fully utilizes the width distinction of the read and write tracks to increase the track density by reducing the width of the write tracks. A write track overlaps part of its neighboring track and the overlapping ensures that it does not affect the

recorded data to be read by the read header. Fig. 2.1 demonstrates the data layout of SMR drive, where the outer tracks are overlapped by the inner tracks, e.g. track 1 is overlapped by track 2.

As the neighboring write tracks are partially overlapped with each other, the write data cannot be directly recorded to the media, as it may overwrite the existing data on its neighboring tracks. Thus only the sequential write is permitted in the SMR drive. To take care of this problem, some research works [17] [14] [29] assemble a certain number of consecutive overlapped tracks into a band, with few non-recordable tracks in-between. With this sort of design, a write request to one band does not affect the data in another band. However, writing to the same band may influence the existing data inside. Thus the entire band needs to be rewritten through 3 steps: reading all the data inside the band, updating them with the new coming data, and writing back the new data into the same band.

The above architecture solves the sequential-only write request problem of SMR drive. Be that as it may, the processing of random write request takes significantly more time compared with the conventional HDD. Some industry players [30] [31] and researchers [32] propose to use a log-structured data recording method to take care of this issue. In this sort of design, a certain number of overlapped tracks are gathered together to form a zone. There is a zone band to isolate the zone with its neighboring zones. The write data, no matter random or sequential, are recorded into the disk zone successively. This introduces the difference between the logical block address (*LBA*) and the recorded physical block address (*PBA*). Thus an address translation mapping is required to record the address translation.

In addition to the address translation, the wastage of disk space is another issue caused by the data update, as the update data are recorded into a new location instead of replacing the old data, which causes the old data to become invalid. The wasted space caused by the invalid data needs to be cleaned to save the disk space.

Figure 2.1: The data layout of SMR drive

GC is one of the strategies to clean the invalid data, which includes the following steps: 1. It needs to choose a suitable zone to do the invalid data cleaning, which can be based on the dirty data ratio, the ratio of invalid data blocks over the total size of the zone. 2. All the valid data in the zone need to be collected in batch and pass to the internal memory buffer. 3. The data gathered should be processed and written to a new zone in sequence.

Some research works have been done to analyze the SMR drive performance under these two different approaches. The SMR drive with log-structured approach is studied through emulation in [33]. The authors use a normal hard disk to emulate a Shingled write disk, which is similar to SMR drive. Simulation is also conducted to study the performance of SMR drive with read-modify-write approach in [34]. A simulation software is designed and developed to analyze the SMR drive performance and disk space utilization. Both of the approaches are controlled at the device level, where the host does not know the address translation information. In addition to the drive level managed SMR drive, there are also host managed SMR drive design, such as [35], whose sequential-only write property is handled at host level, which improves the writing efficiency and minimize the write update operations. Nowadays, researchers [17] [14] [29] add an NAND flash to the SMR drive to improve the performance of data update process under the read-modify-write

approach. The NAND flash is utilized as a cache buffer to keep the write data, which minimizes the number of writing updates to the SMR drive. In [36], the write data is classified into hot data and cold data based on the update frequency, and which are utilized to minimize the data movement in GC process. In [37], a dynamic variable-width striping RAID (DVS-RAID) architecture is proposed for SMR drives, and the proposed method can reduce the updating cost of the parity data. The method handles the data update to the SMR media through appending instead of the read-modify-write approach, which significantly improves the performance of data update process.

2.2 Hybrid Storage Architecture

Hybrid storage systems are categorized as in Fig. 7.2. From the figure, we can notice that there are two types of hybrid storage systems, which are based on tiering method and caching method. Meanwhile, based on these basic architectures, some special storage architectures are built up. For instance, the combination of tiering and caching method, and the HDD device acts as the write cache of the SSD device. In all, there are roughly 4 types of hybrid storage architectures that consist of HDDs and SSDs, which are: 1) SSDs as the cache of HDDs, called caching method, 2) SSDs as the high-end tier to HDDs, which is called tiering method, 3) SSDs as the combination of tier and cache, which is called hybrid method and 4) HDDs with special purposes, e.g., HDDs utilized as the cache or mirror of SSDs. Some research works about the hybrid storage architectures are classified and listed in Table 2.1. There are also some hybrid storage systems incorporating other types of NVMs into design consideration, which will be discussed at the end of this section.

Table 2.1: The literature of hybrid storage systems (HDDs + SSDs)

Categories	Literature
SSD cache	[38] [39] [40] [41] [42] [43] [44] [45] [46] [46] [47] [48] [49] [50] [51] [52] [53] [54] [55] [56] [57] [58] [59] [60] [61] [19] [18] [20] [62] [63] [64] [65] [66] [17] [67] [68] [69] [70] [14] [15] [16] [71] [72] [73] [74] [75] [76] [77]
SSD tier	[78] [79] [80] [81] [82] [83] [84] [85] [86] [87] [88] [89] [90] [91] [92] [93] [94] [95] [96] [91] [97] [98] [99] [100]
SSD as mixed cache and tier	[101] [102] [103] [104] [105] [106]
HDD as the write cache of SSD	[107] [108] [109] [110] [111]

Figure 2.2: The overall categories of the hybrid storage architectures

Table 2.2: Comparison of the hybrid storage systems with SSD caching method

Literature	Cache policy	Data to SSD	Host or drive control	Deduplication	Background data movement	Simulation or protocol
Hot Random offloading [38]	Write back	Random, hot	Host	No	Promote, demote	Protocol
Duplication aware SSD cache [39]	Write back	Hot	Drive	No	Promote	Simulation
SSD array as cache [40]	Write back	Hot read, write	Host	Yes	Flush, promote	Protocol
HFA-hint frequency based [41]	Write back	Hot read, write	Drive, hint	No	Demote, promote	Simulation
MOLAR [42]	Read only	Hot	Drive	No	Promote	Simulation
Service level agreement (SLA) hybrid store [43]	Read only	SLA, hot	Host	No	Promote	Simulation
Lazy adaptive replacement [44]	Write back	Hot	Drive	No	Flush, promote	Simulation
FlashCache [45]	Write back	Hot read, write	Host	No	Flush, promote	Simulation
NAND based disk cache [46]	Write back	Wear leveling aware	Drive	No	Flush, promote	Simulation
Caching less for better performance [47]	Write back	Hot	Drive	No	Flush, promote	Simulation
HEC [48]	Read only	Hot, random	Drive	No	Pre-fetch	Simulation
EDM [49]	Write back	Hot	Host	No	Promote, demote	Protocol
Cache everywhere [50]	Write back	Cost-aware	Drive	No	Promote, demote	Simulation
AMC [51]	Write back	Data value	Drive	No	Promote, demote	Simulation
WEC [52]	Write back	Hot	Drive	No	Promote	Protocol
ETD-Cache [53]	Read only	Hot	Drive	No	Demote	Simulation
Regional popularity aware [54]	Write back	Hot read, write	Drive	No	Promote, Flush	Protocol
NANYANG TECHNOLOGICAL UNIVERSITY SAR [112]	Write back	Hot	Host	No	Promote, demote	Protocol

Figure 2.3: The difference between the host and device controlled policies

2.2.1 SSD Caching Method

To enhance the overall performance of a storage system, SSD is usually utilized as a cache between the DRAM and HDDs to keep the hot read data or the write data [38, 50, 55, 62–65]. Some storage systems with SSD caching methods are shown in Table 2.2. The cache architectures are classified as the host controlled and the device controlled based on the location of the caching methods. In the host controlled caching architecture, the cache algorithm is normally implemented in the host side or operating system, and the data are usually in a file format. In the device controlled cache architecture, the cache policy is usually utilized in the device itself and implemented in the firmware, and the data is usually in block. The differences between the host controlled, and the device controlled policies are shown in Fig. 2.3.

From Table 2.2, we can summarize the common data flows inside the systems as shown in Fig. 2.4. 1. If a read request arrives and its accessing data is located in SSD, the read request can directly access the data from SSD and complete the process. 2. If the accessed data is located in HDD, the data needs to be read from HDD first, and then cached inside the DRAM. If the data is not hot enough, the

Figure 2.4: The data flows in the system with SSD as a cache to HDD

process is completed. 3. If the data accessed is identified as hot by the hot data identification algorithm, the data migration algorithm moves the data to the SSD device. 4. Background data migration can occur for the data identified as hot. 5. The write data can be flushed to the HDD from SSD cache in the background.

Based on the methods to handle the incoming write requests, the storage architectures with SSD caching methods can be categorized into two types: 1. SSD is utilized as the cache for the read request only, which is the read-only policy. 2. SSD can be utilized as the cache for both read and write requests, which is the write-back policy.

2.2.1.1 SSD as a Read-only Cache

In this type of storage architectures [42,43,53,66], when a new write request arrives and its accessing data is not located in SSD, the data needs to be recorded to HDD and the request is completed only when the recording is successful. If the access data is available in SSD, the data in HDD needs to be updated, and the data in SSD may be discarded, or updated as well. Only when both of these updating or discarding processes are successful, the write request is considered as completed. As the write requests are directly passed to HDD in this architecture, the number of write operations to SSD is reduced, and the SSD lifespan is prolonged. Meanwhile,

the cache space utilized by the write data is saved, and the saved cache space can be allocated for the read data, which might improve the read performance. Moreover, the selection of read data to reside in SSD is also important to balance the number of SSD writes and overall system performance. If more data are selected to move into the SSD device, the SSD lifespan is reduced, and more GC processes might be required when the SSD capacity is nearly full. Although the wear-leveling techniques inside the SSD can help to reduce the number of GC processes, it is better to reduce the unnecessary write operations to SSD.

To reduce the unnecessary write operations, [42] proposes a method to check the data hotness based on the demotion counter and the proposed control metric, and migrate the hot data blocks to SSD. In [66], a heuristic file-placement algorithm is designed to improve the cache performance by considering the IO patterns of the incoming workloads. Meanwhile, a distributed caching middleware is implemented at user level to detect and manipulate the frequently accessed data. In [17], a hybrid structure that combines SSD and SMR devices is proposed, which utilizes SSD as the read-only cache to improve the random read performance. In [42], the number of demotions from DRAM to SSD is recorded and utilized to predict the future accesses to SSD, and decide whether the data need to be recorded in SSD or not.

2.2.1.2 SSD as a Write-back Cache

There are two types of write cache, which are write-through cache and write-back cache. In the system with write-through caching policy, the write data is written to the cache and also to the disk, and it is only considered as complete when the data are successfully written to the disk. This policy is mainly utilized in volatile memory cache to avoid the data loss during power off and improve the performance of reading on write data in the cache. Meanwhile, the system with write-back policy can write the data to cache first, and flush the data to disk at a later time, which

can improve the writing performance significantly in some cases. In a hybrid storage system with SSD as a cache, the write-through policy is seldom applied as the SSD is non-volatile memory, which can keep the data safe even when the drive power is lost.

To enhance the writing performance of the hybrid storage system, some research works [63, 67–70] utilize SSD as the cache for both read and write requests. The data operations in the storage architectures with write-back policy are shown in Fig. 2.4. In the architecture, if the write data is new, the data will be recorded to SSD. With the write-back policy, these write data might be flushed to HDD at a later time. However, because of the later flush, the write data in HDD can not be updated simultaneously when it is written to SSD, which may cause data synchronization problem. Period write data flushing might be applied to flush the write data to HDD to improve the data synchronization.

Normally, utilizing SSD as a write-back cache can improve the performance of a hybrid storage system. However, when the SSD is nearly full, GC process is required to clean the invalid data before that the new write requests arrive. This lowers the performance for writing requests, and the overall hybrid system performance might degrade. If the workload is writing intensive with high IOPS, the number of requests to HDD might be large if the re-access ratio is small. Thus the disk may have a limited number of idle periods, and the durations of idle periods will be short. The write data might not be able to be flushed completely to HDD during these short idle periods. Then when the space of SSD is nearly full, the flushing of write data to HDD may further downgrade the system performance. In addition, due to the non-volatility of the SSD device, data can be kept in SSD for a long time without demoting before the space is full. Thus the system performance can be improved by applying the write-back cache policy. However, if the SSD fails, the write data inside might be lost. To overcome the SSD failure problem, the write data inside

SSD is flushed to HDD as soon as possible at the cost of reducing the system performance. Furthermore, the SSD lifespan is another problem as the number of writing operations to SSD is much larger compared with the storage architecture that uses SSD as read-only cache.

There are some research works [14–16], which integrate SSD and the SMR drive as a hybrid storage system to further increase the storage capacity while maintaining the accessing speed. The SSD is utilized as a writing cache to keep the random write data, and leaves the sequential write data to be handled by the SMR drive. At a later time, the random writes are combined and written to the SMR drive in sequence. With this, the amount of random writes to the SMR drive is reduced, as well as the number of read-modify-write operations. The overall performance can be significantly improved.

Industry players apply RAID techniques to SSD cache to improve the reliability of written data. For example, RAID-1 is employed in EnhanceIO [71], RAID-4/5 is employed in LSI ElasticCache [72] and HP SmartCache [73]. In the research area, RAID techniques are also applied to SSDs to improve the data reliability and save the total cost. In [40], a new hybrid storage architecture is designed that employs low-end SSDs to construct a RAID architecture, which is utilized as a cache to improve the data reliability. The new architecture combines the advantages of data reliability of RAID system and performance of the log-structured approach to achieve better performance per dollar and lifetime per dollar compared with high-end SSDs.

Some researchers [46,47] divide the SSD space into read and write regions to improve the system performance with near balanced read and write requests. The read region is only used to store the incoming read data, meanwhile, the write region is only used to record the incoming write data. The data flows for these kinds of storage architectures are shown in Fig. 2.5. As shown in the figure, there are mainly 6 special operations in this architecture: 1. The write data are kept in the write cache

Figure 2.5: The data flows in the system with SSD as a read and write split cache to HDD

region. 2. When a write request needs to access the data located in the read region, there are two options to handle this situation. If the cache is DRAM, the write request will directly update the data located in the read region, and then moves the data from the read region to write region. If the cache is SSD cache, the write request will keep its new data to the write region, and mark the original data in SSD as invalid. The invalid data can be cleaned at a later time. 3. If the data in write region is flushed to the disk, and the data is frequently accessed by read requests, then the clean data can be moved to the read region. 4. The data accessed by read requests can be allocated to the read region. 5. The read data located in the HDD can be moved to the read region if they are identified as hot. 6. The data in the write region can be flushed to disk through background process. With this kind of cache architecture, the cache replacement algorithm can be improved, and the GC process can be more efficient. Thus, the overall system performance is improved. However, if the incoming workload is read-intensive or write-intensive, the corresponding write cache region and read cache region will be wasted, and the overall system performance might degrade.

Write updating to SSD influences the data in the original block and makes them become invalid. GC process is needed to clean these invalid data blocks, which requires extra SSD space. In [47], a new over-provisioned region is split out and

used for the GC process. The split ratios of the read, write and over-provisioned regions are analyzed through modeling to provide an optimal system performance.

The system performance, especially the writing performance, can be enhanced by utilizing SSD as a write-back cache. However, due to the huge number of writes to SSD, the SSD lifespan is reduced significantly. Meanwhile, the number of GC processes required in SSD may also increase. Keeping in mind the end goal to utilize the advantage of SSD write-back cache without reducing the SSD lifespan and increasing the number of GC processes excessively, numerous algorithms [39, 44, 53, 74] and architectures [48] are proposed to reduce the number of writes to SSD. In [39], a duplication-aware write handling strategy is used to filter out the duplicated write requests, which saves the number of writes to the SSD device. In [44], the hot data replacement is done as late as possible, which saves some write operations caused by the data promotion and demotion. Despite the fact that it might miss some new generated hot data, the SSD lifespan is saved and the hot data with longer re-access interval may be kept.

To make the data allocation and migration more precise, the host side information might be utilized to check the hotness of the new coming data, as it might contain more valuable information. For instance, in [75], a multi-level cache hierarchy is proposed and the cache allocation depends on the host side hint information. In the system, the hint information, including access frequency and pattern to different ranges of storage space by each client, is provided to improve the cache performance. Cache partitions for different ranges of storage space are allocated, and the hotter partitions are allocated to the higher speed cache to improve the system performance. Meanwhile, the historical hint information can likewise be utilized to enhance the system performance, e.g., [41] uses historical hint information to enhance the storage system performance. The proposed system consolidates existing works, e.g. Hint-K [113], Demote [114], and Promote [115], together to enhance the

performance of the multi-level cache.

2.2.2 SSD Tiering Method

Tiering techniques divide the data into hot and cold categories, and store them in SSD and HDD, respectively. The data stored in SSD tier is permanent instead of temporary as in the SSD caching, and there is no duplicate copy in HDD. A portion of the research works about SSD tiering strategy are listed in Table 2.3. From the table, we can conclude that the tiered storage architectures can be isolated into two classifications, which are host controlled tiered structure [79,81,82,85,88,93,94] and device controlled tiered structure [78,83,84,86,87,89]. In the host controlled tiered structure, the data movement is controlled by the system host by using the host side information. While the device controlled tiered structure controls the data movement by analyzing the properties of the current incoming data and the historical data.

One of the tiered structures containing SSD tier and HDD tier is shown in Fig. 2.6. The data movements inside the tiered structure include: The data allocation, which is actively performed to designate the data to a suitable tier, which is shown as operations 1 and 2; The data migration, which is performed in background to relocate the data between the two tiers, which is shown as steps 3 and 4 in Fig. 2.6. The data allocation and migration are typically controlled by the host in host controlled tiered structure or the device itself in the device controlled tiered structure.

In the host controlled tiered structure, the choice of which tier to store the approaching data is made by the host through analyzing the host side information. A linearly weighted formulation [82] is applied at the operating system level to characterize the workload access pattern to enhance the overall performance. GreenDM [79] combines the SSD devices and HDD devices in a storage system, and a block level

Table 2.3: Comparison of the hybrid storage systems with SSD tiering method

Literature	Host or drive control	Hot or cold	Read or write	Random or Sequential	SLA and QoS	Data migration	Data allocation	Sim or protocol
Hystor [78]	Drive	Yes	Yes	No	No	Yes	Yes	Protocol
GreenDM [79]	Host	Yes	No	No	No	Yes	Yes	Protocol
Hybrid store [80]	Drive	Yes	No	Yes	No	Yes	Yes	Protocol
Tiera [81]	Host	Yes	No	No	No	Yes	Yes	Protocol
Workload characterization [82]	Host	Yes	Yes	No	No	Yes	Yes	Protocol
E-hash [83]	Drive	Yes	Yes	No	No	Yes	Yes	Sim
Tier warming [84]	Drive	No	No	No	No	Yes	Yes	Both
Extent based dynamic tiering [85]	Host	No	No	Yes	No	Yes	Yes	Protocol
Adaptive data migration [86]	Drive	Yes	No	No	No	Yes	Yes	Protocol
Efficient QoS [87]	Drive	No	Yes	Yes	Yes	No	Yes	Both
Balancing fairness and efficiency [88]	Host	No	No	No	Yes	No	Yes	Both
Automated storage tiering [89]	Drive	Yes	Yes	No	Yes	Yes	Yes	Model
FDTM [90]	Drive	Yes	Yes	Yes	No	Yes	Yes	Sim
Optimal disk allocation [91]	Host	Yes	No	Yes	No	Yes	Yes	Sim

Figure 2.6: The data flows in the system with SSD as a tier

interface is provided to allow data migration between SSD and HDD. GreenDM is adaptable to different kinds of incoming workloads by providing some valuable tunable parameters.

In the device controlled tiered structure, the decision of data movement is made by the device itself, which is the tier module implemented in the firmware, through analyzing the historical access information and the properties of the incoming workload. Hystor [78] analyzes the data access pattern, especially the access locality to find the critical access data. It saves the critical data to the SSD tier to enhance the system performance. Meanwhile, write data is also allocated to the SSD tier to improve the writing performance. Although the writing performance is improved, the SSD lifespan is reduced and the number of GC processes may increase. Hybrid-store [80] utilizes the regional access frequency information through analyzing the historical access information to distinguish the hot data and move them to SSD tier to improve the overall performance.

Data allocation and migration in a tiered storage system are usually determined by the data hotness. Notwithstanding, the device status and the data properties are not considered in these sorts of systems. In [95], a measurement-driven mechanism is applied for the data relocation between different tiers. In the system, the cold data are kept in lower tier. The hot data might be stored to different tiers in view of

the data properties and the status of the devices in various tiers. For instance, the hot random data might be kept in SSD as SSD has faster random accessing speed. The hot sequential accessing data are kept in HDD to save SSD bandwidth as HDD has similar sequential performance compared with SSD. In order to save the SSD lifespan and reduce the number of GC processes in SSD, the hot data identification and data migration algorithms need to be carefully designed. The trade-off between the system performance and the size of written data to SSD needs to be carefully considered.

To save the power consumption of hybrid storage system, [83] develops a novel architecture that consists of one SSD and multiple HDDs, which is called E-HASH. The architecture fully utilizes the advantages of random access of SSD and sequential access of HDDs to improve the system performance while minimizing the overall cost. Meanwhile, the lifespan of SSD is also prolonged.

Tier warm-up is also deployed to improve the tiered storage system performance. In [84], the user access patterns over long time-scales are automatically learned and the intensity of the future user works can be predicted. Based on the knowledge of the tier capacity and performance differences, a schedule can be made to decide when to start and stop the background system works. The data required by the background system works or the user workloads can be pre-filled to the faster tier to improve the overall system performance during the transition period.

2.2.3 SSD Hybrid Method

Both the SSD caching method and SSD tiering method have their own focal points and inconveniences. In tiering method, the frequency of data location optimization is extremely low, usually less than 1 time per hour, or even per day. At the point when the hot data region changes, it needs a considerable measure of time for the

system to move the hot data to SSD to enhance the future system performance. In caching method, the data location is optimized in a short time if the hotness of the data is changed and the system performance in subsequent accesses can be improved. However, the frequent data movements between HDDs and SSDs increase the load of devices, and might influence the overall system performance. To eliminate the restrictions of the caching method and tiering method, the storage architectures in [101, 104–106] combine the tiering method and caching method together. It is the SSD hybrid method that utilizes the SSD as both tier and cache to improve the overall system performance. A simplified data work flow for these kinds of storage architectures are shown in Fig. 2.7: 1. The write data are directly recorded to HDD if the data is not inside SSD. 2. The write request updates the write data in SSD tier region if the data is hot and located in SSD tier. 3. The read request reads data from SSD tier region if the data is located inside SSD. 4. The write request buffers the data to SSD cache region if write-back policy is applied. 5. The read request reads data from HDD and buffers the data to SSD cache. 6. The hot read data is migrated to the SSD tier region in the background. 7. The data movements between the cache and tier regions are optional operations.

[101] consolidates the tiering and caching methods together in a hybrid storage array containing SSDs and HDDs. An allocation proportion is defined to adjust the number of SSDs in the tiering and the caching structure. The system performance, especially for the write intensive workloads is enhanced compared with the one without combination. [102] utilizes the data hotness and the load statuses of the SSD and HDD devices to perform the data allocation and migration. The parallel execution of all the devices are fully utilized to improve the overall performance. In [103], a hybrid storage aware flash translation layer (FDTL) is introduced to allocate the data to different levels of devices. The data allocation is based on the policies of utility maximization and energy consumption minimization. It combines the advantages of low cost and high capacity of HDD, and random access speed

Figure 2.7: The data flows in the system with SSD hybrid methods

and low power usage of SSD to improve the IO performance, reduces the energy consumption and the overall cost.

2.2.4 Other Structures with Special Purposes

Besides storage architectures with SSD as tier and/or cache, there are some other types of storage architectures. For instance, some of the hybrid storage architectures employ HDD as a write cache for SSD [107, 108]. The operations in this kind of storage architecture are shown in Fig. 2.8. Compared with other types of storage architectures, there are 2 special operations: 1. The original data, which is the first copy of data is located to SSD, while the update data will be located to HDD. 2. The write data might need to be flushed to HDD in the background. With this type of architecture, the SSD lifespan is prolonged and the number of GC processes required is significantly reduced as the write update is handled mainly by HDD devices. The only drawback is that some of read requests might need to access both the HDD and SSD to get the complete required data.

I-CASH [107] is a new type of storage architecture that intelligently couples array of SSDs and HDDs. In the architecture, the SSD keeps the write data, called reference blocks, and HDD records the difference between the new coming data and the reference blocks, which is called delta. The read request is handled through the

Figure 2.8: The data flows in the system with HDD as a write cache of SSD

integration of the reference blocks and the deltas, while write request only needs to update the log of deltas, and thus the number of write updates to SSD is minimized. With this architecture, the writing performance is improved, and the SSD lifespan is prolonged. However, the data might be lost if the delta is not saved to HDD properly. Thus, the delta data in the DRAM should be flushed to disk frequently. The flushing interval can be tuned based on the number of write updates that happened in the history.

A hybrid storage design called Griffin [108] uses HDD to cache write data to SSD. The write requests to SSD are logged into the HDD in sequence and migrated back to SSD when the system is not busy. With such kind of design, the number of writes to SSD is reduced and thus the SSD lifespan is prolonged without significantly reducing the read performance.

In [109], the SSD space is split into data zone and delta zone. The data zone is used to store the new coming data, and the delta zone is used to store the difference between the historical data and the new updated data. With the proposed mechanism, the small writing penalty is reduced and the SSD lifespan is prolonged.

To improve the data reliability and synchronization, [110] builds an SSD RAID0 storage system with multiple SSD devices. Moreover, it deploys an HDD with identical space as the mirror to the SSD RAID. The proposed storage system can

improve the data synchronization by lowering the IO performance a bit. In addition, a multi-tier file system (TierFS) [111] is proposed to protect the stored data continuously. The system keeps an extra copy of the updates demoting to the lower tier until the backup of the updated data is completed successfully.

2.2.5 Hybrid Structures Using Various NVMs

As mentioned in Chapter 1, besides NAND Flash, there are other types of NVMs. Some of them have better performance, longer endurance, and lower power consumption compared with NAND. However, their price per GB is much higher than NAND Flash. These kinds of NVMs are often utilized as a storage layer or a cache layer in modern hybrid storage systems to improve the performance cost ratio. In the early stage, NVM is deployed as a cache to the HDD devices [39, 116, 117]. In [117] and [118], the emerging NVM and NAND flash memory are both employed as the cache for HDD to improve the energy efficiency and overall system performance. In [39], NVM is added to improve the power management of the disk through four aspects: 1. extending the idle periods, 2. storing the read-miss content in the cache, 3. spinning up the media only when NVM cannot serve the requests and 4. write throttling. In [119], the potential of PCM in a storage hierarchy is evaluated. The tiering method and caching method are explored by modeling a hypothetical storage system consisting of PCM, HDD, and Flash to offer the best performance within cost constraints by identifying the combinations of different types of devices.

The higher performance NVM is also implemented as a cache for NAND Flash [10–13]. In [10], the data generated by flash translation layer are stored in NVM, which prolongs the Flash memory lifespan. In [12], the combination of DRAM and PCM is deployed as a cache to SSD. A data migration policy is proposed to save the power consumption and enhance the SSD endurance. Meanwhile, the log information of NAND is stored in the PCM [13] to support for in-place updating

and minimize the number of write updates to SSD. It reduces the number of Flash erase operations and prolongs the SSD lifespan.

Tiered memory system that utilizes NVM and DRAM is another type of tiered storage architectures. In [8] and [9], storage class memory (SCM) is deployed together with DRAM to form a tiered memory system. They leverage the advantage of capacity and data loss avoiding when the power failure for SCM, and the advantage of access speed of DRAM. In [120], a new hybrid hardware/software solution that integrates PRAM and DRAM in the storage system is proposed. The write data reliability for PRAM is ensured by applying wear-leveling uniformly through all the PRAM pages by considering the information of writing frequencies.

2.3 Algorithms Utilized in Hybrid Storage Systems

The algorithms and policies applied in the hybrid storage systems actually determine the performance of the overall storage systems. In this section, the most important algorithms, such as data allocation, hot data identification, data migration, and scheduling algorithms are presented. For the easy of representation, we list the most commonly factors considered in these algorithms in Table 2.4, where access frequency and interval are the most important two factors in hot data identification and data migration algorithms. Meanwhile, some algorithms also take the factors in Table 2.5 into consideration, which will not be discussed in detail.

2.3.1 Data Allocation

Hybrid storage system needs to decide which storage devices to handle the new coming data. Especially for the write data, the system needs to allocate the data to

Table 2.4: Two aspects that need to be considered during designing of hot data identification algorithm

Items	Description	Typical algorithms
Access frequency	The number of times a data block is referenced	LFU (least frequently used) [121], GDSF (Greedy-Dual Size Frequency) [122]
Access interval	The time duration that a desired data item being accessed and re-accessed	LRU (least recently used) , MRU (most recently used), LFUDA (LFU with dynamic aging) [121], LRU-K (least recently used k) [123], GDS [124]

proper storage devices to utilize the storage resources. We call this problem as the resource (data) allocation problem [88, 91, 94, 96, 117, 125]. In [91], the data allocation mechanism is formulated as a multiple-choice knapsack problem. The optimal allocation solution can be found through multiple-stage dynamic programming. To maximize the system utilization, bottleneck aware allocation [88] is designed and developed to choose the allocation ratio between clients with the different bottlenecks. To exploit parallelism between the HDD and SSD devices and provide a lower response time, a space allocation policy [96] is designed to find the Wardrop equilibrium by utilizing initial block allocation and migration. Based on the allocation policy, the system can achieve load balancing adaptively across different tiered devices. A smart controller for the disk cache [117] is employed to distribute the IO requests between NAND and PCM, where fine-granularity accesses are directed to the PCM, and coarse-granularity accesses are directed to Flash memory to utilize their advantages.

2.3.2 Address Mapping Algorithms

The address mapping between the SSD cache/tier and the HDD is necessary for the data access in hybrid storage system. Unified address mapping between the logical and physical block groups [103] can be applied to the physical/logical address

Table 2.5: Other workload properties that need to be considered during storage system design

Items	Description
Data size	Generally, only data with small size is required to move to higher tier. The small degree depends on the read/write speed of SSD and HDD, and the migration speed between them, etc. For example, large-size sequential data is usually unnecessary to migrate.
Cache total vs remaining size	The cache size decides how many hot data can be stored in cache. Hence it decides the threshold of hot degree
Device total vs remaining bandwidth	The bandwidth decides if the migration is proper at current time. An approximated function may be built to predict the remaining bandwidth with respect to R/W ratio, IO intensity, etc.
R/W ratio	As the R/W access time and pattern are different, this ratio may give different performance in some scenarios, e.g., the writing amplification and GC in SSD and SMR.
R/W granularity and IO intensity	The value represents the ratio between an IO (both R/W) data amount and the fixed data block size. The average R/W granularity is only considered during certain accessing period. Usually, the data is more important to the user when the value of the data is larger.
Data correlation	One data may be related to another, i.e., Data access to certain data blocks within certain time period may have some access patterns, and other blocks / time periods may have the similar pattern, and hence, they are associated. This value can be used for predication
IO range vs amount vs distribution	IO distribution represents the statistical accessing information, including the accessing address range and the accessing frequency in a given accessing period
Grain size	The minimum size for each page/block to be replaced/migrated
Tier contrast vs compensation (device value)	It values the difference between two different storage tiers/caches for direct data migration, including device status, accessing speed, etc.
Others	Data loss/error, etc

translation. In the system, the address translation map is important and can not be lost, so the translation map is stored into the HDD.

In [126], the addresses of physical data blocks in the secondary storage and addresses of logical blocks corresponding to the physical blocks are stored in the mapping table. Each translation entry contains a value called current level, which is a function of the access frequency and the logarithmic system time. A top-K tracking table with B-tree structure is maintained to keep the first K elements with highest value of current level. When a new mapping entry is created, whose current level will be compared with the lowest one in the top-K tracking table, if it is higher, the lowest one will be replaced by the new coming entry. The value of K is selected based on the memory size allocated to the mapping table.

In [127], the performance difference is considered for different kinds of mapping tables under different run-time or load conditions. An adaptive address mapping with dynamic run-time memory mapping selection method is applied to optimize the mapping performance. A memory controller is deployed to monitor the on-line performance of current memory mapping and dynamically adjust the memory mappings at run-time based on the observed performance.

2.3.3 Hot Data Identification and Data Migration Process

The hybrid system performance is heavily influenced by the selected caching policies, especially the data identification and migration algorithms. Numerous research works have focused on improving the hot data identification accuracy and migration efficiency, some of which are listed in Table 2.6. These hot data identification algorithms are mainly based on factors listed in Tables 2.4 and 2.5.

Table 2.6: Comparison of hot data identification algorithms

Literature	Region levels	Frequ- ency	Recency	Hot levels	Host or drive control	Consider R/W
HFA [41]	One	Yes	No	Hot, cold	Drive	No
Hot random offloading [38]	One	Yes	No	Hot, cold	Host	No
Tiera [81]	One	Yes	Yes	Hot, cold	Host	No
Regional popularity aware cache [54]	Two	Yes	Yes	Hot, cold	Drive	No
Azor [77]	Two	Yes	No	Hot, cold	Drive	Yes
GreenDM [79]	One	Yes	Yes	Hot, cold	Host	No
Multiple bloom filters [97]	Multiple	Yes	Yes	Hot and cold	Drive	Yes
Hotness aware hit [98]	Multiple	Yes	Yes	Hot, warm and cold	Drive	Yes
Block level data valuation [128]	One	Yes	Yes	Specific value	Drive	Yes
Duplication aware cache [39]	One	Yes	Yes	Hot and cold	Drive	Yes
AMC [51]	One	Yes	Yes	Hot and cold	Drive	Yes
HotDataTrap [129]	One	Yes	Yes	Hot and cold	Drive	No
multi-hash-function framework [130]	One	Yes	Yes	Hot, cold	Drive	No

2.3.3.1 Access Frequency

Some hot data identification algorithms [99, 131] identify the frequently used data by counting the amount of access. In [131], the hot data is identified through a probabilistic modeling of the incoming data, which can reduce the ratio of false detection of cold data as hot data.

2.3.3.2 Access Frequency and Recency

Besides the access frequency, some algorithms [39, 86, 90, 97, 132] also utilize the access recency, which is last accessing time of the data to identify the hot data. Temperature aware caching (TAC) [62] checks the hotness of the data by analyzing the temperature statistics, which is the access frequency and the age of each region. A bloom filter [97] is applied to check the hotness of data through spatial locality and temporal locality. HotDataTrap [132] tries statistical approach to avoid some cold data being inserted into the cache, which gives more chance for the hot data to be stored in the cache space. An improved adaptive cache replacement algorithm named D-ARC [39] is proposed to identify the hot data through both the recency and frequency of the incoming workload. A D-ARC algorithm can improve the cache hit ratio, reduce average I/O latency and prolong the SSD lifespan significantly for certain types of workloads. Multiple independent hash functions [130] are applied to increase the correctness of hot data identification.

2.3.3.3 Frequency/Recency with Other Factors

Some algorithms [38, 128] use other properties of the incoming workload to identify and migrate the hot data to SSD. Hot Random Off-loading method [38] checks the randomness and hotness of incoming data to determine the direction of the file migration. In [128], a block-level valuation model is applied with the data value

viewing from file-level to classify the data in the storage. In [133], a framework is proposed to detect and predict the properties changing in the workload to make the migration process automatically. The detection and prediction are done through a clustering method utilizing a Markov chain correlation algorithm.

To save the power consumption of tiered storage system, [134] proposes a power-aware multi-level cache policy, which is used to identify and select data for migration. In a storage system, the SLA is quite important for the Quality of Service (*QoS*) requirement by the user. How to explore the available storage resources to provide high *QoS* to customers is a quite challenging problem. In [135], a new approach is presented to meet different kinds of SLA requirements through data migration processes in varied environments. The data allocation and migration are decided by the SLA requirements and the temperature of the data, which is the hotness of the data.

2.3.3.4 Data Migration Algorithms

Together with the data hotness, the data migration algorithms [51, 100, 134, 136, 137] also need to consider other factors, such as the incoming workload properties and the statuses of the devices. To improve the storage and retrieval performance, in [100], a metric is created to categorize all the levels of storage pools, which is applied to prioritize the data segments, and the data migration is performed based on the priority of the segments. In [136], the data migration policy is designed based on the data value computed through data intrinsic attributes and the prospective values. In [86], the migration deadline is also considered, and an adaptive data migration model is proposed to attempt to keep the hot data in SSD. The adaptive data migration is realized through IO profiling and look ahead migration where the IO profiling predicts and exploits the hot accessed area and the look-ahead migration moves the data that has potential to become hot in near future to SSD. In [138], offline

algorithms are explored for the Flash cache by considering both the hit ratio and Flash lifespan. A multi-stage heuristic is designed to approximate the offline optimal algorithm and provide the same optimal read hit ratio while reducing the number of Flash erases significantly. In [137], a bi-directional migration policy is applied to balance the storage QoS and migration cost. The policy has two thresholds to determine the data promotion and demotion. In [90], a fully automatic migration process is proposed and a double thresholds method is introduced to only migrate the most suitable data blocks. The data migration can be done in two directions: low-end to high-end and high-end to low-end. An adaptive multi-level cache (AMC) replacement algorithm [51] is applied to adjust the cache blocks between different cache levels. When new data arrives, AMC can dynamically determine the cache level to locate the incoming data by combining the selective promotion and demotion operations. With AMC, the caching resource management becomes more efficient, and multi-level cache exclusiveness is achieved.

2.3.4 Cache Scheduling Algorithms

2.3.4.1 Cache Scheduling Algorithms in HDD

There are numerous types of scheduling algorithms utilized in the HDD device to optimize the disk service time, including the seek time, rotational latency and data transfer time, such as Rotation Position Optimization (RPO) [139], Shortest Access Time First (SATF) [140], and SCAN [141], etc. However, due to the fundamental difference between SSD and mechanical driven HDD, these conventional scheduling algorithms implemented in HDD are not suitable for SSD drive and hybrid systems.

2.3.4.2 Cache Scheduling Algorithms in SSD

To fully utilize the SSD properties, an efficient scheduling scheme [142] for Native Command Queuing (*NCQ*) in SSDs is proposed to improve the system performance. In the scheme, the I/O service time for each command is estimated based on the status of the buffer as well as the physical characteristics of SSDs. The scheduler selects the next command to be executed based on the estimated service time. Some scheduling algorithms also consider the workload properties. In [143], a workload-aware budget compensation scheduling algorithm is utilized for the device-level request scheduling. The scheduler estimates the cost of GC contribution for each of the virtual machines in the SSD device and compensates the budget for each virtual machine based on the estimated cost for performance isolation.

2.3.4.3 Cache Scheduling Algorithms in Hybrid Storage Systems

Based on the difference of the scheduling algorithms in HDD and SSD, new scheduling algorithms need to consider both of the properties of HDDs and SSDs. The combination of HDD scheduling and SSD scheduling algorithms can be one of the solutions for hybrid storage system. In [144], a hybrid disk-aware Completely Fair Queuing (*CFQ*) is proposed to optimize the I/O reordering in new hybrid storage systems. In [145], a Parallel Issue Queuing (*PIQ*) scheduling method is applied to increase the parallel execution of incoming requests. It attempts to combine the non-conflict requests into the same batch to avoid unnecessary access conflicts.

Scheduling algorithms may also consider the SLA from users. To guarantee the SLA, [146] proposes a scheduling algorithm to provide guaranteed service level objectives in a dynamic environment. The proposed algorithm can dynamically adjust the number of active hosts, which makes the system suitable for various kinds of workloads with different IOPS and throughput, while minimizing the overall system

power consumption.

Chapter 3

Theoretical analysis of SMR drive performance

3.1 Introduction

From the literature related to the SMR drive design, we notice that there are three types of SMR drives, host managed SMR drive, host aware SMR drive and device managed SMR drive. The device managed SMR drive design can be utilized as the storage device similarly compared with conventional HDD devices. Currently, there are two approaches to deal with the data recording to the device managed SMR drives: read-modify-write approach and log-structured recording approach.

However, there are still issues that should be solved in both of the SMR drive designs. In the design of read-modify-write, the average response time for the write requests is substantially longer than conventional drive. It is on the fact of that the write request to the band of SMR drive may require three steps to complete, which includes reading the data from the band, modifying the data, and writing

back the data to the band. In spite of the fact that the design of log structured write takes care of the issue of large response time of writing request, more invalid data are generated and the disk capacity is wasted. GC process can be utilized to clean these invalid data in the background. However, when the workload coming to the disk is heavy, the idle time periods might be insufficient for the GC process. The amount of dirty data might keep increasing if they can not be cleaned on time. The performance of the SMR drive turns out to be worse when the disk capacity is nearly fully utilized, as the disk needs to clean some of these dirty data to provide enough space for handling a new coming write request.

To tackle the issue, in this chapter, a new GC policy is proposed to solve the poor performance of SMR drive when the disk capacity usage reaches the limit. In the new strategy, we classify the GC process as background GC process and foreground GC process. Background GC process only cleans the invalid data during the disk idle periods, which has a lower priority compared with user requests. Foreground GC process cleans the invalid data together with user requests no matter the disk is in idle mode or not, and it has same priority as user requests. The background GC and foreground GC processes are selected based on the changing disk status, including idle time, disk fullness, etc. The bandwidth allocation for the foreground GC is also determined dynamically based on the disk status. With the new proposed GC policy, the performance of SMR drive turns out to be more predictable, particularly when the disk capacity usage nearly reaches the limit.

However, with the plenty number of parameters and policies for dealing with the write requests and GC processes, and the changing requirements from the client, it is not easy to compare and evaluate the effectiveness of these algorithms, especially the performance of SMR drive under these designs in practice. Some simulation tools are designed and developed to conduct the comparison and evaluation, however, the selection of algorithms and tuning of drive parameters are still difficult through

try-and-error approach. Theoretical analysis and modeling are required to provide a guideline and validation for simulation and the actual drive design. In this chapter, we utilize a theoretical approach to study the performance of the disk mechanical component statistically under *RPO* scheduling algorithm, including the seek, rotation and transfer time. An M/G/1/K queue with state dependent service time is proposed to analyze the disk performance and solved through a Markov Chain model. By applying the proposed model, we can get the disk blocking probability under different request arrival rates, workload environments and drive settings, and estimate the maximum IOPS that the SMR drive design can support. Meanwhile, a simulation tool is developed to validate the analytical modeling. The results from simulation and analytical modeling cross-validate with each other and provide acceptable performance difference.

The main contributions of this chapter are as follows:

- We formulate the SMR drive as an M/G/1/K queue with state dependent service time, and the formulation can be utilized to analyze the performance of SMR drive under different workload environments and drive setting parameters.
- We design and develop an SMR drive simulator to simulate the drive performance under different settings to validate the analytical model, the simulation results show the correctness of the model. A new GC policy is proposed to solve the existing significant performance degradation problem when disk capacity is nearly full.
- We study and optimize the SMR drive performance under different drive settings, such as write batch size, GC batch write size and disk status, etc., and under different workload environment, such as read ratio, request size and sequential ratio, etc., through theoretical and simulation approaches.

The remainder of this chapter is organized as follows. Section 3.2 discusses about the historical research works about the SMR drive and some analytical models related to the drive performance analysis. Section 3.3 formulates the performance of the SMR drive as a queuing model problem. Section 3.4 analyzes the SMR drive performance through queuing theory. Section 3.5 shows the details of the simulation tool designed for the SMR drive. Section 3.6 compares and analyzes the performance of the SMR drive under analytical modeling and simulation. Finally, Section 3.7 draws the conclusion.

3.2 Background

In Chapter 2, several SMR driver research works have been analyzed, most of these SMR drive performance analysis and designs are through simulation, emulation and prototype. None of them studies the performance of the SMR drive through a theoretical approach. In order to build an analytical model to study the SMR drive performance, it is better to know the analytical approach for conventional storage devices. Historically, the performance of HDD device is studied by lots of researchers [147] [148] [149] [150]. Due to the different data recording properties and request processing method of SMR drive, such as the batch write and garbage collection method, etc., these models can not be directly applied to the modern SMR drive design. However, some basic disk operations, such as disk scheduling algorithm and queue modeling, etc. can still be applied as they have similar features.

Due to the mechanical properties of the HDD, the seek and rotational distance affects the overall drive performance a lot. Instead of processing disk requests in coming order, reordering these requests based on the seek distance or rotational distance can significantly improve the drive performance. These kinds of request reordering mechanisms are called disk scheduling algorithms. SCAN algorithm [141]

processes the queuing requests based on the disk seek distance, which minimizes the overall seek time. There are some mathematical formulations constructed to analyze the performance of SCAN and Circular SCAN (C-SCAN) scheduling algorithms. Besides the overhead of the seek time, the rotational latency is another factor that affects the overall disk performance. Shortest Access Time First (SATF) [140] [151] is introduced to address this problem, which reorders the requests in the queue based on their processing time, including seek time, rotational latency and data transfer time. The performance of the disk drive under different scheduling algorithms is studied and evaluated in [140] and [152]. Nowadays, industry players normally apply the RPO scheduling [139] to reorder the requests in the queue, which optimizes the disk seek time and rotational latency.

Queue behavior is another important factor to be considered when analyzing the disk performance, especially when the workload is heavy and the disk is busy. As the size of the memory buffer in the disk is always limited, the disk queue is normally modeled with limited queue capacity, such as $M/M/1/K$ and $M/G/1/K$ [153], etc. In $M/G/1/K$ model, G is disk state independent service time distribution, which is not suitable for SATF, SCAN, and RPO scheduling algorithm, as the disk service time is dependent on the number of requests in the queue. Historically, $M/G/1$ [154] [155] and $M/G/1/K$ [155] queues with state dependent service time have been studied and analyzed in different areas. Meanwhile, [147] proposes to use $M/G/1/K$ queue with different service rate to study the performance of HDD with the SATF scheduling algorithm.

However, there is no research work which utilizes them to analyze the performance of an SMR drive. To fill the gap of analytical modeling for the SMR drive, in this chapter, a theoretical approach is proposed to evaluate the SMR drive performance. $M/G/1/K$ queue with state dependent service rate is utilized to model disk processing mechanism of the SMR drive with the RPO scheduling algorithm.

Figure 3.1: The illustrative diagram for request queuing in SMR drive with GC process

3.3 Problem Formulation

In an SMR drive, the shingled recording property does not allow the data to be recorded randomly. Meanwhile, the speed of sequential access for several requests is much faster compared with random access of the same requests, as the seeking and rotation time are saved. Thus, it is possible to combine the random write requests to the SMR drive into a sequential one, which fulfills the requirement of shingled write property of SMR drive, as well as improves the write performance of the SMR drive. In other words, it improves the overall throughput, and IOPS for the heavy random write workloads compared with conventional methods.

Although the sequential write can improve the SMR drive performance, the sequential processing of update requests may cause existing data to become invalid. The invalid data wastes the disk space, which is required to be cleaned to save the disk capacity. GC process is introduced to clean the invalid data.

Fig. 3.1 shows the mechanism of an SMR drive, including the sequential write

process and the GC process. In the figure, the external requests are user requests, which are issued from host side, with pre-defined arrival rate λ . When requests arrive at the external queue, the read request will be directly passed to the internal queue if the data is not cached. Meanwhile, the write request will be buffered in the external queue. If the number of write requests in the buffered queue reaches a pre-defined value, which is the write batch size, then all the buffered write requests will be combined as one write request, and passed to the internal queue. In the internal buffer, there are two types of operations. One is the user read request and the batched write request, and the other is the GC operation, which include the GC read operation and GC write operation. As shown in Fig. 3.1, zone 1 is a normal zone containing data for coming read request, zone 3 has the largest ratio of dirty data, zone $X + 1$ is the current active zone for keeping write data. The disk can only process one of these operations at one time. Let us denote $\lambda_1, \lambda_2, \lambda_3$ as the IOPS and T_1, T_2, T_3 as the service time for the normal read, batch write, and GC process, respectively. Let us also denote the request size, read request ratio and sequential ratio of the incoming workload as S_{req}, r_r and r_s . And denote the write batch size as b_1 , GC batch write size as b_2 and disk fullness status as s_{disk} , which is the capacity used over the total capacity of the disk. There are other parameters and settings, which are listed in Table. 3.1.

Then we formulate the SMR drive in terms of the followings: (a) workload property formulation; (b) disk processing time; (c) Disk constraints explanation; and (d) performance evaluation.

3.3.1 Workload Property Formulation

Each request has an arrival time that records the time slot the request arrives at the external buffer of the SMR drive. The time interval between the two adjacent requests is called inter-arrival time. In a normal workload environment, the inter-

Table 3.1: SMR drive and input workload related parameters

Name	Description	Default value
r_r	Read request percentage	0.4
S_{req}	Average request size	32K
r_s	Host request sequential ratio	0.0
b_1	Host write batch size	8
b_2	GC read batch size	8
r_{gt}	Garbage ratio threshold	0.3
t_{idle}	Idle time threshold	0.005s
s_{disk}	Disk fullness ratio	0.8
r_{dirty}	Average zone garbage ratio	0.3
$p_b^{threshold}$	Queuing block ratio threshold	0.1%

arrival times are not constant, and here, we use the Poisson distribution to represent the inter-arrival time distribution. Let us denote the request arrival rate as λ , which is the number of requests that arrive at the external queue in a certain period. Thus $1/\lambda$ is the expected inter-arrival time.

The external workload may contain a different number of read requests and write requests, and the ratio of the read requests may vary under different workload environments. Here, we denote the read request ratio as r_r , which is the number of read requests over the total number of requests. Thus the arrival rate of read requests $\lambda_1 = r_r \lambda$, and the arrival rate of write requests is $(1 - r_r) \lambda$.

The requests in the external workload may have the different sizes, which is the number of data blocks the requests want to access the disk. Here, we use S_{req} to denote the average request size of all the requests.

The sequential access requests mean that two or more requests that access the data in a sequential order. For example, if the last access data block of request A is blk_1 and the first data access block of request B is blk_2 , and $blk_1 + 1 = blk_2$, then the request A and B are considered as sequential access requests. The number of sequential access requests over the number of total requests is the sequential ratio of

the workload which is denoted as r_s . In the internal queue, the sequential accessed requests can be combined together as one request, which increases the expected request size to S_{req}/r_s , and reduces the arrival rate to λr_s .

3.3.2 Disk Processing Time Formulation

Table 3.2: SMR drive setting parameters

Name	Description	Value
n_C	Cylinder number	350786
n_S	Surface number	1
n_{SPT}	average sector per track	2050
S_{sec}	sector size	512
S_{seg}	total track number (size) per segment	(default 1)
s_{RPM}	rotation per minute	7200
t_f	full revolution time	8.33ms

To analyze the overall performance of the SMR drive, firstly we need to formulate the disk processing time for the normal user request and the GC operation T_1 , T_2 and T_3 , which are formulated as (3.1) and (3.2). The detailed explanation of these formulations can be found in [156]. In the equations, $t_{pos}^{(n)}$ and t_{tran} are analyzed in detail in Section 3.4, and their calculations are based on the parameters of the drive mechanical components, which are shown in Table 3.2.

$$\begin{cases} T_1 = t_{pos}^{(n)} + t_{tran} \\ T_2 = t_{pos}^{(n)} + b_1 t_{tran} \\ T_3 = t_{gc}^r + t_{gc}^w \end{cases} \quad (3.1)$$

In the equation, the $t_{pos}^{(n)} = t_{seek}^{(n)} + t_{rot}^{(n)}$, and the t_{gc}^r and t_{gc}^w are expressed as follows:

$$\begin{cases} t_{gc}^r = t_{pos}^{(n)} + b_2 t_{tran} \\ t_{gc}^w = t_{pos}^{(n)} + b_2(1 - r_{dirty})t_{tran}. \end{cases} \quad (3.2)$$

3.3.3 Disk Resource Constraints

In an SMR drive, there are two constraints that affect the drive performance: internal queue length and the disk capacity usage ratio. In this section, we will explain these two constraints in detail.

3.3.3.1 Internal Queue Length Constraint

As the buffer of the queue is finite due to the hardware limitation, the number of requests can be buffered in the queue is limited also. Let us denote the capacity of the external and internal queues as K_1 and K_2 , which are the maximum numbers of requests allowed to be buffered in the external and internal queues. In current HDD design, the media controller can only handle one request at a time. If the disk is busy handling one request, the other coming requests need to wait for the current one to be completed. And these requests will be buffered in the external queue or internal queue during the waiting time.

If the external queue and internal queue are both full, and there is a new request coming, the new coming request will be blocked. The blocked request may cause data loss or increase the response time, which may downgrade the user experience. In order to limit the blocking request to occur too frequently, we define a blocking probability threshold, denoted as $P_b^{threshold}$ to match the user QoS requirement. If $P_b < P_b^{threshold}$, we consider that the incoming user workload requirement can be fulfilled by the SMR drive design. Otherwise, the user needs to change the workload or the design of the SMR drive. Different users may require different QoS for the SMR drive design, such as the bank transaction may require $P_b^{threshold} = 0$, video storage may require $P_b^{threshold} = 0.5\%$, etc. For simplicity, we apply $P_b^{threshold} = 0.1\%$ as an illustration. +

Figure 3.2: The method to select different GC policies

3.3.3.2 Disk Capacity Usage Constraint

As shown previously, the SMR drive uses shingled recording technique, and the data is recorded in disk sequentially. The invalid data caused by write update wastes the disk space. When the disk capacity is nearly full, these invalid data need to be cleaned to give space for new coming write requests. Disk processing bandwidth needs to be allocated to the GC process to clean the invalid data, especially when disk is nearly full, e.g. $s_{disk} > 0.9$, and the number of idle periods is not enough for GC process. The allocation of disk bandwidth to GC process significantly reduces the overall performance.

To solve the inconsistent performance of SMR drive when disk capacity is nearly full, a GC policy selection algorithm is proposed as shown in Fig. 3.2. In the flowchart, the "Idle time detection process" is utilized to check whether the disk is in idle mode. It compares the time duration that the internal queue length is equal to 0 and the idle time threshold, which is pre-defined by the drive to check the disk idleness. The "Require foreground GC module" mainly checks the disk fullness status s_{disk} and the disk capacity cleaned by the background GC. If s_{disk} is large, and the amount

of data cleaned by the background GC is not enough, then the foreground GC is required to help for the data cleaning. Otherwise, the foreground GC is deactivated to avoid the performance degradation for user requests.

If the disk has lots of idle time periods, the foreground GC is not activated, only background GC is performed, which does not reduce the performance λ_{max} significantly. When the disk keeps in busy state, a certain amount of disk bandwidth is allocated to the foreground GC based on the disk dirty ratio and the s_{disk} , which reduces the λ_{max} performance. However, this allows the disk to start cleaning the invalid data earlier to avoid significant performance loss when $s_{disk} > 0.9$. When the disk only has few idle time periods, both foreground GC and background GC are activated to clean the invalid data, which can further reduce the impact on the λ_{max} performance, while maintaining the speed of invalid data cleaning.

In the proposed policy, when s_{disk} is large, e.g. $s_{disk} > 0.6$ and there are not enough idle periods, we need to allocate certain percentages of disk processing bandwidth to the foreground GC process. The amount of disk processing bandwidth allocated to foreground GC process increases when s_{disk} increases. With this constraint, the disk processing bandwidth allocated to the user requests reduces.

3.3.4 Performance Evaluation

To evaluate the performance of an SMR drive, there are a lot of performance metrics that can be constructed, such as the disk capacity usage, overall response time, user requests ratio, throughput and IOPS, etc. In this chapter, we use the maximum IOPS the SMR drive can support, denoted as λ_{max} under various types of workloads to study the SMR drive performances. By providing the workload properties, such as r_r , r_s and S_{req} , etc., and the SMR drive parameters, such as b_1 , b_2 , s_{disk} and t_{idle} , etc., the arriving rate λ is λ_{max} , when the following two conditions are met:

a). $P_b < P_b^{threshold}$; b). There is sufficient disk processing bandwidth allocated to the GC process for cleaning the dirty data.

3.4 Analysis

As shown in the previous section, the performance metric λ_{max} is determined by two factors: the requests blocking ratio in the internal queue P_b , and the disk fullness status s_{disk} . In this section, we will calculate the P_b theoretically and estimate the effects of s_{disk} on the drive performance. The method of estimating the λ_{max} will also be provided.

3.4.1 Disk Processing Time Analysis

To evaluate λ_{max} under different workloads and drive parameter settings, the first important thing to know is the basic disk performance metric, which is the disk processing time. The disk processing time consists of 3 components: seek time, rotational latency and data transfer time. The time distributions of these components are different based on different scheduling algorithms. In the RPO scheduling method [140], the seek time and rotational latency are related to the number of requests in the disk queue, while the data transfer time is only affected by the size of data to be transferred. For simplicity, we assume that the expected data transfer time is only related to the average request size, which can be expressed as:

$$E[t_{tran}] = E[S_{req}] \times \frac{60}{RPM \times SPT}, \quad (3.3)$$

where SPT means sectors per track.

3.4.1.1 Seek Time Distribution

In current disk model, the number of sectors per track increases from inner track to the outer track. There are more data can be stored on the outer track compared with inner track. For simplicity, we assume all the tracks have an equal number of sectors, which is equal to the number of sectors on the middle track in the actual drive. Seek time is the time usage for the disk head to move from the source track Tr_s to the destination track Tr_d . The distance of the seek S_d can be expressed as $S_d = |Tr_s - Tr_d|$. To calculate the seek time distribution, we should know the relationship among the seek time, seek distance and the seek distance distribution.

Lemma 3.4.1. *Assume all the tracks have an equal number of sectors, which is equal to the number of sectors on the middle track in the actual drive. Based on the disk property [149], if the minimal seek time and maximum seek time are denoted as t_{min}^{seek} and t_{max}^{seek} respectively, and the total number of tracks is denoted as Tr_{total} , then the probability density function (PDF) of the seek time can be expressed as (3.4), and the cumulative distribution function (CDF) of the seek time can be expressed as (3.5).*

$$f_{seek}(t) = \begin{cases} 2/u, & 0 \leq t < m + n \\ \frac{4(t-m)}{u} - \frac{2(t-m)^3}{n^2 u^2} & otherwise \end{cases} \quad (3.4)$$

$$F_{seek}(t) = \begin{cases} 0, & 0 \leq t < m + n \\ \frac{2(t-m)^2}{n^2 u} - \frac{(t-m)^4}{n^4 u^2} & otherwise \end{cases} \quad (3.5)$$

where $u = Tr_{total} - 1$, $m = \frac{t_{min}^{seek} \sqrt{u} - t_{max}^{seek}}{\sqrt{u} - 1}$ and $n = \frac{t_{max}^{seek} - t_{min}^{seek}}{\sqrt{u} - 1}$.

Proof. According to [157], the seek time is related to the seek distance, and the relationship is expressed as (3.6). In the equation, the maximum seek time is expressed

as t_{max}^{seek} , and the minimal seek time as t_{min}^{seek} , which is the seek time for one track. The total number of tracks is expressed as Tr_{total} and let $u = Tr_{total} - 1$, which is the maximum seek distance the disk head can perform.

$$t_{seek}(S_d) = \begin{cases} 0, & S_d = 0 \\ \frac{t_{min}^{seek}\sqrt{u}-t_{max}^{seek}}{\sqrt{u-1}} + \frac{t_{max}^{seek}-t_{min}^{seek}}{\sqrt{u-1}}\sqrt{S_d} & otherwise \end{cases} \quad (3.6)$$

As we assume all of the tracks have same amount of sectors, they have the same probability to be accessed by a random request, and the PDF of the track Tr being accessed can be expressed as:

$$f_{Tr}(x) = \frac{1}{u} \quad where \quad u = Tr_{total} - 1. \quad (3.7)$$

The PDF of the seek distance is calculated by combining all the possible combinations of two distinct tracks, denoted as Tr_1 and Tr_2 . During computation, there are two cases to be taken care of: 1) $Tr_1 > Tr_2$, and 2) $Tr_1 \leq Tr_2$. The distribution function is expressed as:

$$\begin{aligned} f_{dis}(x) = & \int_0^{u-x} f_{Tr}(y)f_{Tr}(y+x)dy \\ & + \int_x^u f_{Tr}(y)f_{Tr}(y)f_{Tr}(y-x)dy. \end{aligned} \quad (3.8)$$

After some mathematical simplification, we can get the expression of f_{dis} as:

$$f_{dis}(x) = \frac{2(u-x)}{u^2}. \quad (3.9)$$

With the PDF expression, the CDF of the seek distance F_{dis} can be expressed as:

$$F_{dis}(x) = \frac{2x}{u} - \frac{x^2}{u^2}. \quad (3.10)$$

Then we can apply the CDF and PDF of seek distance $F_{dis}(x)$ and $f_{dis}(x)$ to get the PDF and CDF of the seek time based on [157], which can be expressed as:

$$F_{seek}(t) = \begin{cases} F_{dis}(0), & 0 \leq t < m + n \\ F_{dis}\left(\left(\frac{t-m}{n}\right)^2\right) & otherwise, \end{cases} \quad (3.11)$$

and

$$f_{seek}(t) = \begin{cases} f_{dis}(0), & 0 \leq t < m + n \\ f_{dis}\left(\left(\frac{t-m}{n}\right)^2\right) 2(t-m) & otherwise. \end{cases} \quad (3.12)$$

By substituting (3.9) and (3.10) into (3.12) and (3.11), we can get (3.4) and (3.5). $\square$

Similar model can be applied to the zoned disk model, which has a larger number of sectors on the outer tracks and a smaller number of sectors on the inner tracks. The distribution of the track being accessed can be formulated as a linear increasing model, with increasing rate of $(sec[Tr_{total} - 1] - sec[0]) / (Tr_{total} - 1)$. Then the seek distance distribution and seek time distribution can be computed accordingly.

3.4.1.2 Rotational Latency Distribution

Let us denote $f_{rot}(x)$ as the rotational time distribution, where x is the angle required to be rotated to the target sector. As the disk rotation speed is constant with pre-specified RPM, we can express $f_{rot}(x)$ as a uniform distribution, with a minimal value of 0, and maximum value of R_{max} , which is the time required to rotate for the full track. Then the distribution of $f_{rot}(x)$ can be expressed as:

$$f_{rot}(x) = 1/R_{max} \quad 0 \leq x \leq R_{max}, \quad (3.13)$$

and the CDF of rotational latency is:

$$F_{rot}(x) = x/R_{max} \quad 0 \leq x \leq R_{max}. \quad (3.14)$$

3.4.1.3 Disk Processing Time under Different Queue Length

The seek time and rotational latency distribution functions are given in the previous section, which are $F_{seek}(x)$, $F_{rot}(t)$, $f_{seek}(t)$ and $f_{rot}(x)$. Now we can apply the PDF of the seek time and rotational latency to compute the PDF of disk positioning time $f_{pos}(x)$, which is the convolution of the seek time and rotational latency.

Lemma 3.4.2. *Given the PDF of the seek time and rotational latency as (3.4) and (3.13), according to the properties of the RPO scheduling algorithm [140], the PDF of disk positioning time and service time can be expressed as (3.15) and (3.16).*

$$f_{pos}(x) = \begin{cases} \frac{1}{R_{max}} \int_0^x f_{seek}(u) du, & 0 \leq u < R_{max} \\ \frac{1}{R_{max}} \int_{x-R_{max}}^x f_{seek}(u) du & R_{max} \leq u < t_{max}^{seek} \\ \frac{1}{R_{max}} \int_{x-R_{max}}^{t_{max}^{seek}} f_{seek}(u) du & u \geq t_{max}^{seek} \end{cases} \quad (3.15)$$

$$f_{service}(x) = \begin{cases} 0, & 0 \leq x < t_{tran} \\ f_{pos}(x - t_{tran}) & otherwise \end{cases} \quad (3.16)$$

Proof. With known $f_{seek}(x)$ and $f_{rot}(x)$, we can get the disk positioning time by the convolution of the disk seek time and rotational latency, which is expressed as (3.17).

$$f_{pos}(x) = f_{seek+rot}(x) = \int_{-\infty}^{\infty} f_{seek}(x-z)f_{rot}(z)dz \quad (3.17)$$

By providing the limitation for (3.17) with lower bound as 0, and upper bound as R_{max} , we can further express (3.17) as:

$$\begin{aligned} f_{pos}(x) &= \int_0^{R_{max}} f_{seek}(x-z)f_{rot}(z)dz \\ &= \frac{1}{R_{max}} \int_0^{R_{max}} f_{seek}(x-z)dz \\ &= \frac{1}{R_{max}} \int_{x-R_{max}}^x f_{seek}(y)dy \quad \text{where } y = x-z \end{aligned} \quad (3.18)$$

As the disk seek time is in the range $[0, t_{max}^{seek}]$, the y in (3.18) must be in the seek time range. Thus (3.18) needs to be calculated in piecewise, which are ranged as $[0, R_{max})$, $[R_{max}, t_{max}^{seek})$, and $[t_{max}^{seek}, R_{max} + t_{max}^{seek}]$. After some algebraic calculations, we can get the disk positioning time distribution as shown in (3.15). Meanwhile, by adding the disk data transfer time into the equation, which gives us the service time distribution as (3.16).

□

By using the results from Lemma 3.4.2, we can get the CDF of the positioning time $F_{pos}(x)$ and the disk service time $F_{service}(x)$ by integrating the $f_{pos}(x)$ and $f_{service}(x)$, i.e.

$$F_{pos}(x) = \int_0^x f_{pos}(z)dz \quad (3.19)$$

$$F_{service}(x) = \int_0^x f_{service}(z)dz \quad (3.20)$$

Lemma 3.4.3. *If the disk applying the RPO scheduling algorithm, and the PDF and CDF of disk positioning time are known as (3.15) and (3.19), then we can get the PDF of the minimal disk positioning time under different queue lengths through (3.21).*

$$f_{pos}^{(1)}(x) = i(1 - F_{pos}(x))^{i-1} f_{pos}(x), \quad (3.21)$$

where i is the number of requests in the queue, and $f_{pos}^{(1)}(x)$ is the PDF of smallest disk positioning time of all the requests in the disk queue.

Proof. In the disk with the RPO scheduling algorithm, the disk service time is the minimal disk positioning time for all the queuing requests plus the disk transfer time, which is dependent on the number of sectors to be transferred. Thus, the disk service time can be expressed generally as:

$$t_{service}^i = \min_{r=1,\dots,i} (t_{pos}^r) + t_{tran}. \quad (3.22)$$

In the equation, t_{pos}^r is the disk positioning time, which includes the seek time and rotational latency. t_{tran} is the disk transfer time under different request size. To calculate the distribution of the $t_{service}^i$, we need to know the minimal positioning time distribution. Here, order statistics [158] is employed to calculate the distribution of the minimal positioning time by using the first order distribution. Assume there are i requests in the disk queue, denoted as $R_1, R_2, \dots$ and R_i , which are independent and identically distributed. Then the CDF of the minimal positioning time can be expressed as:

$$F_{pos}^{(1)}(x) = 1 - (1 - F_{pos}(x))^i \quad (3.23)$$

3.3.1: CDF

3.3.2: PDF

Figure 3.3: The CDF and PDF of disk positioning times under different queue lengths.

From (3.23), we can get the PDF of disk positioning time, by getting the derivation of the CDF of the disk positioning time distribution $F_{pos}^{(1)}(x)$, which is (3.21).

□

Similar to the results from Lemma 3.4.3, the PDF of disk service time can be expressed as:

$$f_{service}^{(1)}(x) = i(1 - F_{service}(x))^{i-1} f_{service}(x) \quad (3.24)$$

Based on (3.21) and (3.24), we can compute the $f_{pos}^{(1)}(x)$ and $f_{service}^{(1)}(x)$ numerically. For instance, we assume the maximum rotational latency R_{max} to be $8.33ms$, which is based on the $RPM = 7200$, and t_{max}^{seek} to be $12.96ms$. Then we can get the PDF and CDF of disk positioning time as shown in Fig. 3.3.

3.4.2 P_b Calculation and M/G/1/K Queue Analysis

In an SMR drive applying the RPO scheduling algorithm, the disk service time is dependent on the number of requests in the queue, and the queue length distribution

3.4.1: Positioning time

3.4.2: Queue length

Figure 3.4: The disk positioning times and number of requests in disk queue under different IOPS.

is related to the disk processing time and the request arrival rate. It is not possible to use traditional M/G/1/K queue analysis to find a solution of P_b , as the service time G is state dependent.

In this section, we apply a theoretical approach to find out the queue length distribution and the blocking probability by given the request arrival rate λ and the disk service time $f_{service}^{(1)}(x)$ under different queue lengths.

Lemma 3.4.4. *The disk queue length state transition matrix P is irreducible and aperiodic.*

Proof. From [159], we know that a Markov chain is said to be irreducible if it is possible to get to any state from any state. Let us assume that state i and j are a pair of states in P , $i = j$ means the state does not change. This could happen when the disk queue only receives one request during the disk processing time. If $i > j$, this could occur when the disk queue does not receive any request during $i - j$ disk processing times. Thus we can say that P is irreducible. To prove that P is aperiodic, first let us define the periodic of a state.

$$k = \gcd(n > 0 : \Pr(X_n = i | X_0 = i) > 0), \quad (3.25)$$

where gcd is the greatest common divisor. If $k > 1$, then the state is periodic and if $k = 1$, then the state is aperiodic. From [159], we know that 'A Markov chain is aperiodic if every state is aperiodic. Let us assume that there is a state i in P . If $i = 0$, which means that the state of the queue length is 0, then the state can be reached from state $j = 1$, when there is no request arriving at the queue during the disk processing time. If $i = K$, the state can be reached from $j = K - 1$, when there are 2 requests that arrive during the disk processing time. For any other state, the state can be reached from $j = i - 1$ when there are 2 requests, or from $j = i + 1$ when there is no request arriving during the disk processing time. Thus we can say that the state in P is aperiodic, and the Markov chain matrix P is aperiodic. $\square$

Theorem 3.4.1. *If the PDF of disk service time is given as $f_{service}^{(1)}(x)$ in (3.24), the disk arrival rate is λ , the disk queue buffer size limitation is fixed as K , and assume that the request processing in the queue follows a Markov Chain model, then the queue length distribution at departure instance exists and can be obtained by solving (3.26).*

$$\begin{cases} \pi^{(d)} = \pi^{(d)} P \\ \sum_{L=0}^{K-1} \pi_L^{(d)} = 1 \end{cases} \quad (3.26)$$

Proof. Let t_n denote the n - th departure instant, which means a request is just leaving the queue buffer and being serviced by the disk. Let L_n denote the queue length at t_n , then we have the following condition:

$$0 \leq L_n \leq K - 1, \quad (3.27)$$

where K is the maximum capacity of the queue, which is the maximum number of requests that the queue can buffer. Let α_n denote the number of requests that arrive during $[t_{n-1}, t_n)$. Assume L_n is known at the time instance, then we can obtain the distribution of α_{n+1} through:

$$Pr(a_{n+1} = k | L_n) = \int_0^\infty f_{T|L_n}(t) \frac{(\lambda t)^k}{k!} e^{-\lambda t} dt, \quad (3.28)$$

where $f_{T|L_n}(t)$ is the PDF of disk service time under queue length L_n . With α_{n+1} , we have the following relation:

$$L_{n+1} = \min\{L_n - 1 + \alpha_{n+1}, K - 1\}. \quad (3.29)$$

Based on (3.29), we can observe that the sequence of L_n , $n = 1, 2, \dots, K - 1$ is a Markov chain. Let $P = (P_{L,L'})$ be the transition probability matrix, in which $P_{L,L'}$ is the transition probability from queue length L to queue length L' . Based on (3.28), we can get $P_{L,L'}$ through the following equation:

$$P = \begin{bmatrix} \alpha_{0,1} & \alpha_{1,1} & \alpha_{2,1} & \dots & \alpha_{K-2,1} & 1 - \sum_{n=0}^{K-2} \alpha_{n,1} \\ \alpha_{0,1} & \alpha_{1,1} & \alpha_{2,1} & \dots & \alpha_{K-2,1} & 1 - \sum_{n=0}^{K-2} \alpha_{n,1} \\ 0 & \alpha_{0,2} & \alpha_{1,2} & \dots & \alpha_{K-3,2} & 1 - \sum_{n=0}^{K-3} \alpha_{n,2} \\ 0 & 0 & \alpha_{0,3} & \dots & \alpha_{K-4,3} & 1 - \sum_{n=0}^{K-4} \alpha_{n,3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & \alpha_{0,K-1} & 1 - \alpha_{0,K-1} \end{bmatrix}.$$

In order to know whether there exists an equilibrium queue length distribution, and the chain can converge to the steady state, we need to check whether the transition matrix P is irreducible and aperiodic. By applying Lemma 3.4.4, we can prove that P is irreducible and aperiodic, then there exists an equilibrium distribution. Let us denote the steady state queue length distribution at departure instant as $\pi^{(d)}$, and the steady-state probability of queue length L as $\pi_L^{(d)}$. Then we can have (3.26).

By solving these linear equations, we can get the steady-state queue length distribution at departure instance, which is $\pi^{(d)}$. $\square$

With the steady-state distribution $\pi^{(d)}$ at departure instance gotten from (3.26), we can calculate the average service time $E[T]$ and the average queue length $E[Q]$ through (3.30) and (3.31).

$$E[T] = \sum_{L=0}^{K-1} (E[T|L] \times \pi_L^{(d)}) \quad (3.30)$$

$$E[Q]^{(d)} = \sum_{L=0}^{K-1} (L \times \pi_L^{(d)}) \quad (3.31)$$

Lemma 3.4.5. *If the queue length distribution at the departure instance is $\pi_L^{(d)}$, then the request blocking ratio can be calculated through (3.32).*

$$\pi^{(a)}(K) = \frac{1 - \sum_{j=0}^{K-2} \pi^d(j)}{\frac{1}{\pi_F^{(a)}(1)} - \sum_{j=0}^{K-2} \pi^d(j)}, \quad (3.32)$$

$$\pi_F^{(a)}(1) = \frac{\lambda}{\lambda + \frac{1}{E[T]}}, \quad (3.33)$$

$$\pi_F^{(a)}(0) = \frac{\frac{1}{E[T]}}{\lambda + \frac{1}{E[T]}}, \quad (3.34)$$

where $\pi_F^{(a)}(0)$ and $\pi_F^{(a)}(1)$ are the probability distributions that there is 0 or 1 request in the queue.

Proof. As we have:

$$\pi^{(a)}(i) = \pi^{(d)}(i)(1 - \pi^{(a)}(K)), \quad i = 0, 1, \dots, K-1, \quad (3.35)$$

and:

$$\pi^{(a)}(K) = (1 - F_F(K-2))\pi_F^{(a)}(1), \quad (3.36)$$

where:

$$F_F(i) = \sum_{j=0}^i \pi^{(a)}(j). \quad (3.37)$$

Through (3.37), we can get $F_F(K-2) = \sum_{j=0}^{K-2} \pi^{(a)}(j)$ in (3.36). Based on (3.35) and (3.36), we can get (3.32) through some algebraic calculation. $\square$

From (3.32), we can get that the probability of that the queue length is K , which means the disk queue buffer is full. When the disk queue is in this state, the new coming request will be blocked. Thus $\pi^{(a)}(K)$ is the disk blocking probability with an arrival rate λ . We can compute the disk request blocking probability P_b through Algorithm 1. In addition, the average queue lengths and average disk processing times under different λ can also be calculated through (3.30) and (3.31). The comparisons of the theoretical and simulation results of the expected queue length and the disk service time are shown in Fig. 3.4. From the figure, we can observe that the differences between the analytical results and simulation results are less than 10%. These small differences do not affect the drive performance analysis, especially the performance trends under various kinds of parameter settings and workload environments.

Based on Algorithm 1, we can get the P_b under different settings and request arrival rates. Then we can use binary-search algorithm to find the maximum IOPS that the SMR drive can support, while keeping the P_b under restriction, which is $P_b < P_b^{threshold}$. This maximum IOPS is the λ_{max} we are searching for.

3.4.3 Performance Modeling under Different User Batch Write Sizes

Since we mainly focus on the status of the internal queue to check the P_b , it is required to gather the properties of the requests entering or buffering in the internal queue. The requests include the read request, batch write request, GC read and GC write request. In this part, we only consider the SMR drive performance without GC process, so the main incoming requests are the read request and the batch write request.

As the write requests in the external queue are batched together as one request to

Algorithm 1 Blocking probability calculation

Input: $\lambda_0, K, f_{T_0|m}(t)$
Output: P_b

```

1: Initialization :
2:  $P = (P_{L,L'}) = 0$ 
3: Compute  $Pr(\alpha|m)$  :
4: for  $m = 1$  to  $K - 1$  do
5:   for  $i = 0$  to  $K - 1$  do
6:      $Pr(\alpha = i|m) = \int_0^\infty f_{T_0|m}(t) \frac{(\lambda_0 t)^i}{i!} e^{-\lambda_0 t} dt$ 
7:   end for
8: end for
9: Compute  $P = (P_{L,L'})$  :
10: for  $L_1 = 1$  to  $K - 1$  do
11:   for  $L'_1 = L_1 - 1$  to  $K - 2$  do
12:      $P_{L_1,L'_1} = Pr(\alpha = L'_1 - L_1 + 1|L_1)$ 
13:   end for
14:    $P_{L_1,L'_1} = 1 - \sum_{n=0}^{K-L_1-1} Pr(\alpha = n|L_1)$ 
15: end for
16: Compute  $P_b$  :
17:  $\pi = \pi P$ 
18: for  $i = 1$  to  $K - 1$  do
19:    $p_i^{(d)} = p_i$ 
20: end for
21:  $E[T] = \sum_{m=0}^K (E[T|m] \times p_i^{(d)})$ 
22:  $\pi_F^{(a)}(1) = \frac{\lambda}{\lambda + 1/E[T]}$ 
23:  $F_F(K - 2) = \sum_{j=0}^{K-2} \pi^{(d)}(j)$ 
24:  $\pi^{(a)}(K) = \frac{1 - F_F(K-2)}{1/\pi_F^{(a)}(1) - F_F(K-2)}$ 
25: for  $i = 1$  to  $K - 1$  do
26:    $\pi^{(a)}(i) = \pi^{(d)}(i)(1 - \pi^{(a)}(K))$ 
27: end for
28:  $P_b = \pi^{(a)}(K)$ 
29: return  $P_b$ 

```

the internal queue, the IOPS and the request size for the internal queue are different from the external queue.

Lemma 3.4.6. *If the request arrival rate to the external queue follows a Poisson distribution, the arrival rate to the internal queue will also follow a Poisson distribution.*

Proof. From [160], we know that the sum of two Poisson processes with arrival rates of λ_a and λ_b will still be a Poisson process with an arrival rate $\lambda_a + \lambda_b$. Meanwhile, splitting of a Poisson process with an arrival rate of λ will generate two or more Poisson process with an arrival rate of $\alpha\lambda$ and $\beta\lambda$.

If the workload has an arrival rate λ , and read ratio r_r , then the read and write requests will follow Poisson distributions with an arrival rate of $\lambda_1 = r_r\lambda$ and $(1 - r_r)\lambda$, respectively. The batch write request to the internal queue will also follow a Poisson distribution with an arrival rate $\lambda_2 = (1 - r_r)\lambda/b_1$. The combination of read and batch write requests to the disk internal queue also follows a Poisson distribution with an arrival rate of $\lambda_1 + \lambda_2$, which is $\lambda(r_r + (1 - r_r)/b_1)$. $\square$

With the request arrival rate at the internal disk queue, the expected size of requests to the internal buffer queue S_{act} can be expressed as:

$$S_{act} = S_{req}/(r_r + (1 - r_r)/b_1). \quad (3.38)$$

Since there is no GC process, we can assume $\lambda_3 = 0$, then the arrival rate at the internal queue λ_{act} can be expressed as:

$$\begin{aligned} \lambda_{act} &= \lambda_1 + \lambda_2 + \lambda_3 \\ &= \lambda r_r + \lambda(1 - r_r)/b_1. \end{aligned} \quad (3.39)$$

Through binary searching, the maximum IOPS λ_{act} at the internal queue can be found, we can utilize the searching result λ_{act} to compute the λ_{max} with (3.39).

$$\lambda_{max} = \lambda_{act} / (r_r + (1 - r_r) / b_1) \quad (3.40)$$

3.4.4 Performance Modeling under Different GC Batch Write Sizes

As shown in the previous section, there are two types of GC process: foreground GC process and background GC process. The background GC is only conducted when the disk is idle, and it does not affect the performance of the user requests a lot. The impact can be minimized to nearly 0 if the background GC process can be canceled during the GC process being handled by the disk. In order to analyze the λ_{max} under different GC batch write sizes b_2 , we focus on the foreground GC process. It has the same priority as user requests, and it is performed only when the disk fullness status s_{disk} is large enough. If the arrival rate of the foreground GC process also follows a Poisson distribution, then the combination of GC processes and user requests also follows a Poisson distribution, which can be proved similarly as Lemma 3.4.6.

Theorem 3.4.2. *Based on the request processing model utilized in the SMR drive [31], the average request size to the disk internal queue S_{act} can be expressed as (3.41) when the disk is in foreground GC mode.*

$$S_{act} = \frac{S_{req}(1 + b_2^{\frac{(2-r_{dirty})}{2}}(r_r + \frac{1-r_r}{b_1}))\frac{b_{gc}}{1-b_{gc}}}{r_r + \frac{1-r_r}{b_1} + (r_r + \frac{1-r_r}{b_1})\frac{b_{gc}}{1-b_{gc}}} \quad (3.41)$$

Proof. Assume λ is the arrival rate of the user requests, then the user request arrival rate to the internal queue is:

$$\begin{aligned}\lambda_{internal} &= \lambda_1 + \lambda_2 \\ &= \lambda \left(r_r + \frac{1 - r_r}{b_1} \right).\end{aligned}\tag{3.42}$$

The arrival rate of GC processes will be:

$$\lambda_3 = \lambda \frac{\left(r_r + \frac{1 - r_r}{b_1} \right) b_{gc}}{1 - b_{gc}}.\tag{3.43}$$

The total amount of data being processed in the time duration is:

$$S_{total} = S_{req} \left(\lambda + \lambda_{gc} \frac{b_2 + b_2(1 - r_{dirty})}{2} \right).\tag{3.44}$$

Then the average size of the requests to the internal queue S_{act} can be calculated through:

$$S_{act} = \frac{S_{total}}{\lambda_{internal} + \lambda_{gc}}.\tag{3.45}$$

With some algebraic calculations, we can get the expression in (3.41). $\square$

The arrival rate of the requests to the internal queue λ_{act} can be expressed as:

$$\begin{aligned}\lambda_{act} &= \lambda_1 + \lambda_2 + \lambda_3 \\ &= \frac{\lambda r_r + \lambda(1 - r_r)/b_1}{1 - b_{gc}}.\end{aligned}\tag{3.46}$$

Through binary searching, we can get the maximum IOPS at the internal queue λ_{act} . Then we can compute the λ_{max} with the (3.46).

$$\lambda_{max} = \frac{\lambda_{act}(1 - b_{gc})}{r_r + (1 - r_r)/b_1}\tag{3.47}$$

3.4.5 Performance Modeling under Different Disk Fullness Statuses

Under different s_{disk} , the policy selection for the GC process is different. If s_{disk} is less than a pre-defined threshold, such as 0.5, the disk does not require the foreground GC process as the disk capacity is still enough for the future write requests. So the performance evaluation can be conducted through equations (3.38), (3.39) and (3.40). When s_{disk} is larger than the pre-defined threshold, foreground GC process is required to clean the invalid data. The disk processing bandwidth b_{gc} is calculated based on s_{disk} , and it increases when s_{disk} increases. The performance of the SMR drive under this scenario is estimated through equations (3.41), (3.46) and (3.47).

3.5 Simulation Setup

To validate the proposed analytical model, we design and develop an SMR drive simulator to simulate the performance of the SMR drive under different drive settings and workload environments. The overall structure of the simulator is shown in Fig. 3.5. There are mainly 5 components in the SMR drive simulator: 1. Trace generator; 2. SMR handler; 3. Scheduler; 4. Disk access controller; and 5. Disk model.

Workload generator generates user workload based on different workload settings, such as request size, read request ratio, sequential ratio and arrival rate, etc.

SMR handler handles the user requests and the GC process. The external buffer queue is located at this level. The user read request is directly passed to the next module for disk processing. Meanwhile, the user write request will be buffered in the external queue. When the number of write requests in the external queue reaches the write batch size b_1 , all the write requests will be combined and passed to next

Figure 3.5: The overall structure of the SMR drive simulator

module. The GC process is also initiated by this module. When the disk condition, such as disk capacity usage and idleness, etc. meets the GC process requirements, GC read and GC write operations will be issued to the disk. Meanwhile, GC policy selection algorithm is also implemented in this module.

Scheduler maintains the disk internal queue, and buffers all the requests, including user requests and GC processes. The RPO scheduling algorithm is implemented in this module to decide which specular request to be handled by the subsequent disk operation. In the queue, priority is given to different kinds of requests based on the drive design. For example, lower priority is given to the background GC process compared with user requests. The same priority is given to the foreground GC process and user requests.

Disk processes the incoming request and calculates the seeking time, rotational latency and data transfer time for the incoming request.

3.6 Results and Discussion

From the analytical model, we observe that the λ_{max} performance of the SMR drive is mostly affected by the parameters: write batch size, GC batch write size and the disk status, which are used to allocate disk processing bandwidth to the GC process

and normal user requests. The performance of the SMR drive also varies under different workload properties, such as the read request ratio, average request size and sequential ratio. It is important to evaluate the performances of SMR drives under these settings and workload environments to provide hints for future SMR drive design. Thus in this section, we study the SMR drive performance by using the analytical model and the simulation tool with same settings and workloads to cross-validate each other. And we also study the performance trends of the SMR drive under different settings and workload environments.

3.6.1 Performance Analysis under Different User Write Batch Sizes

To study the SMR drive performances under different write batch size settings for different kinds of workloads, we fix other drive parameters as shown in Table 3.1. The simulation results and the analytical results are shown in Fig. 3.6. Fig. 3.6.1 shows the λ_{max} performances under different read request ratios. Fig. 3.6.3 and Fig. 3.6.4 show the λ_{max} and throughput performances under different request sizes, and Fig. 3.6.2 shows the λ_{max} performance under different sequential ratios. These figures show that the differences between the theoretical analysis and the simulation results are less than 15%.

As shown in Fig. 3.6.1, the λ_{max} performance degrades when the r_r increases. This is because of that the number of write requests decreases when r_r increases. Then the benefit of the batch write to combine several random write requests into one sequential write request is reduced, and the reduction of request arrival rate to the internal queue is less, thus the overall performance degrades with increasing r_r . Additionally, we also notice that λ_{max} is larger when b_1 is larger with fixed r_r . This is a consequence of the IOPS to the internal queue that can be further reduced

3.6.1: read ratio

3.6.2: sequential ratio

3.6.3: request size

3.6.4: request size throughput

Figure 3.6: SMR drive performance under different workload properties and write batch sizes.

by increasing b_1 . Fig. 3.6.1 also shows that the increasing of b_1 can significantly improve λ_{max} when r_r is small. However, the performance improvement is not so significant when r_r is large, as the performance of the read request is not affected by the batch process. Although increasing b_1 can improve the overall performance when r_r is small, we can not increase b_1 without a limitation. This is because larger b_1 means that more memory buffer space needs to be allocated for the write requests, and less memory space is available for caching the read data.

Fig. 3.6.3 exhibits the results for the SMR drive performance under different request sizes. It is obvious that λ_{max} decreases with increased request size S_{req} , as more data transfer time is required for handling the requests with larger S_{req} . With a similar disk positioning time, the overall disk service time is larger for a request with larger S_{req} , which reduces the overall disk service rate and λ_{max} . Although λ_{max} is smaller when S_{req} is larger, the overall throughput is larger compared with the workload that has the smaller S_{req} , which can be found in Fig. 3.6.4.

The performance λ_{max} under different sequential ratios r_s is also studied, and the results are presented in Fig. 3.6.2. Under different b_1 , λ_{max} always increases with increasing r_s , as the disk can save a significant amount of positioning time to handle the sequential requests, which increases the disk service rate. We can also notice that the λ_{max} improvement with increasing b_1 becomes less significant when r_s increases. The main reason for this is that the sequential requests have similar performance no matter whether applying or not applying the batch process, while random requests with the batch process have better performance.

3.7.1: read ratio

3.7.2: sequential ratio

3.7.3: request size

3.7.4: request size throughput

Figure 3.7: SMR drive performance under different workload properties and GC batch write sizes.

3.6.2 Performance Analysis under Different GC Batch

Write Sizes

Similar to the SMR drive performances study under different write batch sizes, the performances of SMR drive under different GC batch write sizes are also studied with a similar manner by fixing other parameters as shown in Table 3.1. The simulation results and the analytical results are shown in Fig. 3.7. In which, Fig. 3.7.1 shows the λ_{max} performances under different read request ratios, Fig. 3.7.3 and Fig. 3.7.4 show the λ_{max} and throughput performances under different request sizes and Fig. 3.7.2 shows the λ_{max} performances under different sequential ratios.

Fig. 3.7.1 shows a similar performance changing pattern as in Fig. 3.6.1. The only difference between them is the value of λ_{max} , which is mainly caused by the foreground GC process handled by the SMR drive, requiring a certain amount of the disk processing bandwidth (In this experiment, we use 20% of disk process bandwidth for GC process). The performances λ_{max} under different GC batch write sizes b_2 are also illustrated. It is easy to notice that λ_{max} decreases with increased b_2 , as the amount of data needed to be cleaned by GC process increases with larger b_2 , which requires more data transfer time, and thus the overall disk processing time. However, the data cleaning speed can increase, which is the benefit of larger b_2 . It is especially useful with larger s_{disk} .

In Fig. 3.7.3, we notice that λ_{max} degrades when the request size S_{req} increases. A similar performance changing pattern can be found in Fig. 3.6.3. This is on the fact that the requests with larger S_{req} always require longer data transfer time for different values of b_1 and b_2 , which increases the disk service time and decreases the disk processing rate.

As shown in Fig. 3.7.2, λ_{max} increases with increasing r_s . The reason can be illustrated similarly as the one shown in Section 3.6.1. The main reason behind this

3.8.1: read ratio

3.8.2: sequential ratio

3.8.3: request size

3.8.4: request size throughput

Figure 3.8: SMR drive performance under different workload properties and disk statuses.

is that requests with larger r_s save the disk processing bandwidth and increase the disk service rate, which increases the λ_{max} . Meanwhile, we also notice that the λ_{max} performance degrades when b_2 increases. This is similar to the one shown in Fig. 3.7.1, as both of them need more time to handle the GC process with larger b_2 , which obviously reduces the disk service rate for user requests and decreases λ_{max} performance.

3.6.3 Performance Analysis under Different Disk Fullness Statuses

Similar study is also conducted on the performance of SMR drive under different disk statuses, where other drive parameters are fixed as shown in Table 3.1. The simulation results and the analytical results are shown in Fig. 3.8, where Fig. 3.8.1 shows the λ_{max} performances under different read request ratios, Fig. 3.8.3 and Fig. 3.8.4 show the λ_{max} and throughput performances under different request sizes and Fig. 3.8.2 shows the λ_{max} performances under different sequential ratios.

Fig. 3.8.1 shows a similar performance changing pattern as Fig. 3.6.1 with different r_r . The differences between them are the actual values of λ_{max} under different settings and the performance trends under different b_1 and s_{disk} . In Fig. 3.8.1, the actual values of λ_{max} are much smaller than the ones shown in Fig. 3.6.1 with same r_r settings, as a certain amount of disk processing bandwidth is allocated to the GC process in Fig. 3.8.1, which lowers the service rate for user requests and thus λ_{max} . With increasing s_{disk} , more disk processing bandwidth is allocated to GC process, and λ_{max} will further decrease. As a result, the overall trends of λ_{max} decrease with increasing s_{disk} .

Similar to Fig. 3.6.3 and Fig. 3.7.3, Fig. 3.8.3 also shows that the λ_{max} decreases and throughput increases with larger S_{req} . The λ_{max} performance degradation with increased s_{disk} can be found in Fig. 3.6.3, which is caused by the same reason as shown in Fig. 3.8.1.

The performance increment with increasing r_s under different s_{disk} can be illustrated by Fig. 3.8.2. The reason for this is similar as the one we explained in Section 3.6.1.

Based on the results shown in Fig. 3.8.1, 3.8.3, and 3.8.2, we can conclude that s_{disk} only affects the value of the λ_{max} , but has minimal impact on the performance

changing trends under different workload properties, such as the read ratio r_r , request size S_{req} and sequential ratio r_s . Meanwhile, λ_{max} always reduces when s_{disk} increases. This can be explained by that the larger disk processing bandwidth is allocated to the GC process when s_{disk} increases, which reduces the disk service rate for user requests. However, since we start the foreground GC process earlier, such as $s_{disk} = 0.5$, more dirty space can be cleaned through these foreground GC processes before that the disk status reaches capacity limit, e.g. $s_{disk} > 0.9$. Although λ_{max} performance reduces a bit, which is dependent on the bandwidth allocation policy. However, the speed of disk capacity usage reduces and the dirty ratio of the whole disk decreases, which means the disk usage ratio increases. More time is allowed for the disk to be utilized before $s_{disk} = 0.9$ is reached. As the overall dirty ratio reduces significantly, even when $s_{disk} > 0.9$ is reached, the priority of foreground GC is not so high compared with normal SMR drive design, which may have a large dirty ratio. Thus the performance of SMR drive is improved when $s_{disk} > 0.9$, and the speed of s_{disk} reaching to full is lower compared with the one without applying foreground GC policy.

In all, we can notice that the performance differences between the simulation results and the analytical results are less than 15% for all the disk parameters and workload properties. In some cases, the performance error is even less than 10%. The performance trends under different workload parameters and drive settings are also similar. This allows us to use the analytical modeling to model the SMR drive and provide reasonable performance estimation under different drive designs and workload environments. It can also work as a guide for the future SMR drive design.

3.7 Conclusion

This chapter provides an analytical model and a simulation tool to study the SMR drive performance under different kinds of workload environments, such as read ratio, request size and sequential ratio, etc., and SMR drive setting parameters, such as write batch size, GC batch write size, and disk fullness status, etc. We analyze the disk processing time and queue length distribution with RPO scheduling algorithm under different arrival rates through an M/G/1/K queue modeling with state dependent service time. The λ_{max} is also studied through the generated model based on the user QoS requirement, such as request blocking ratio. A new GC policy selection algorithm is proposed to solve the poor performance of SMR drive when the usage of disk capacity reaches the limit. The results from analytical modeling and simulation tool are studied and compared, which cross-validate the correctness of the model and the simulation tool. The 10%-15% performance difference between the analytical and simulation results allows us to use the analytical modeling as a guide for the future SMR drive design.

From the simulation results, we notice that there are still gaps between the analytical and simulation results, the gaps might be minimized by the following three approaches. Firstly, the distribution of the incoming requests can be generated more closely to the theoretical model to have a closer performance. However, the number of requests to be generated must be large enough, which increases the simulation time to 10x or even 100x. Secondly, the driver overhead can be taken into consideration in the analytical model. This will make sure that the analytical model is more realistic, however, the complexity of the analytical model will be significantly increased. Thirdly, a more realistic data layout model can be applied in the analytical model. Note that the actual data layout in real devices has more data sectors in the outer track and fewer data sectors in the inner track. However, the

complexity of the analytical model will be increased, and the input trace data need to be more random to cover the inner, middle and outer tracks to have a reasonable distribution. These enhancements will be further investigated in future.

Chapter 4

Optimal Selection of GC Dirty Threshold to Balance Energy Consumption and Space Release

4.1 Introduction

From the previous chapter, we note that there are two typical designs for the SMR drives, read-modify-write approach and log-structured recording approach. We also know that the log-structured recording approach is commonly applied for the SMR drive design in industry. GC process becomes important to the design as the invalid data needs to be cleaned to fully utilize the drive space. Meanwhile, the performance of the SMR drive with log-structured recording approach is studied under different kinds of workloads and drive settings.

From the previous chapter, we know that the decision of the SMR drive to make GC process to a zone or not is based on the garbage ratio of the zone, which is the

number of dirty data blocks over the total number of data blocks in a zone.

However, it is not easy to balance the capacity usage and energy consumption. If the zone selected for GC process has small garbage ratio, the benefit of GC process to that zone may not cover the cost of energy consumption, as the number of dirty data blocks cleaned is limited. Thus there is one important drive parameter called garbage ratio threshold to determine whether to perform GC process or not. The selection of garbage ratio threshold affects the overall cost saving, which is the benefit of capacity saving minus the cost of energy consumption. If the garbage ratio threshold is too large, a lot of invalid data will not be able to be cleaned and the disk space can not be used efficiently. If the garbage ratio threshold is too small, the cost, including energy consumption and performance degradation, to clean the invalid data is larger compared to the benefit of disk space saving. Thus an efficient method to tune the garbage ratio threshold based on the workload properties and drive parameters is important to the overall cost reduction of the drive.

In this chapter, the garbage ratio threshold selection problem is formulated as an optimization problem based on the cost of energy consumption and benefit of capacity saving. Different GC policies are studied based on the formulated problem. Simulations are conducted to verify the cost of energy consumption and benefit of disk capacity cleaning.

The main contributions of this chapter are as follows:

- An optimization problem is formulated to find the most suitable garbage ratio threshold for the GC process under different workload properties and drive parameters. The formulation is done based on the disk capacity cleaned, energy consumption and the performance degradation caused by the GC process.
- A simulator for the SMR drive is designed and developed to study the performance and energy consumption of the SMR drive. The cost savings under

different settings can be estimated with the simulator.

- The cost savings of the SMR drive with different garbage ratio threshold settings under different scenarios are evaluated based on the analytical and simulation results, which can be used as a guideline for the SMR drive design.

The remainder of this chapter is organized as follows. Section 4.2 discusses the related works about the SMR drive. Section 4.3 describes the problem formulation for the selection of GC dirty ratio threshold based on the cost of energy consumption and benefit of disk capacity saving. Section 4.4 evaluates the impacts of selecting different GC dirty ratio thresholds on the overall cost savings under different drive parameters and workload environments. Finally, Section 4.5 draws the conclusion.

4.2 Background

The performance of SMR drive has been investigated by many researchers, which are discussed in Chapter 2. These research works simulate or emulate the performance of SMR drives with read-modify-write or log-structured approaches. Both of these approaches are trying to solve the sequential write only property of the SMR drive while maintaining the write performance of the SMR drive. To improve the write latency problem of an SMR drive with the read-modify-write approach to handle the update data, an NAND Flash is added to the SMR drive in [17] [14] [29] to keep the update data. This design tries to avoid the number of write updates to the disk media. Thus the cost of data update to the SMR drive is minimized and the overall performance of write requests is improved. To minimize the number of write updates to the drive media, a dynamic variable-width striping RAID (DVS-RAID) architecture is introduced by [37] to reduce the overhead of updating parity data to the media. It appends the parity data to the media instead of read-modify-write approach. Although the GC process is required to clean the dirty data caused

Figure 4.1: A feed-forward controller to control the GC dirty ratio threshold

by the sequential appending, the performance of the overall system is significantly improved. Slightly different from the drive-managed SMR disks, the host managed SMR disks may have more IO pattern information. In [35], the write and update requests are handled at the host level. With the host-side information, the data placement can be handled more efficiently and the number of read-modify-write operations can be minimized with the host-level management. However, the host level management requires the modification of the host level operating system, which does not allow the SMR drive to be used interchangeably with conventional HDD drive.

Although the performance of SMR drive is studied in many works, the power consumption of the SMR drive is still not investigated in detail, especially the power consumption caused by different kinds of GC processes. In this chapter, the power consumption and disk capacity cleaning under different kinds of GC processes are studied and compared.

4.3 Problem Formulation

In an SMR drive, GC garbage ratio threshold, denoted as r_{dirty}^t is important to decide whether the GC process is necessary for the disk zones that contain dirty data. To effectively select the most suitable r_{dirty}^t , a feed-forward controller, as shown in Fig. 4.1, is constructed with the workload properties and drive parameters

as the controller inputs.

In the controller, the r_{dirty}^t calculator is conducted through an optimization process. The process takes the workload properties and drive parameters as the inputs, and finds out a r_{dirty}^t that can provide maximum cost saving, which can be estimated through the equation (4.1). Table 4.1 lists out the necessary parameters utilized in the formulation.

Table 4.1: The parameters utilized in the problem formulation and simulation

Name	Description	Default value
r_r	Read request percentage	0.4
S_{req}	Average request size	32K
r_c	Background GC cancellation ratio	0
b_1	Host write batch size	8
b_2	GC read batch size	8
r_{dirty}^t	Garbage ratio threshold	0.3
t_{idle}	Idle time threshold	0.005s
s_{disk}	Disk fullness ratio	0.8
P_{seek}	Seeking power for read and write	$0.1 - 15w$
P_{rot}	Rotation power for read and write	$0 - 6.0517w$
P_{tran}	Data transfer power	$0 - 6.0517w$

$$\min \quad \beta P_c + \beta_1(\lambda_{max} - \lambda_{max}^{gc}) - \alpha C_s \quad (4.1)$$

where β, β_1, α are the coefficients for the cost of energy consumption, performance degradation, and capacity cleaned. P_c is the energy consumption, C_s is the total size of data blocks being cleaned, λ_{max} is the maximum IOPS the drive can support without GC process, and λ_{max}^{gc} is the maximum IOPS the drive can support with GC process.

The constraint conditions for (4.1) are different under different GC policies, which are varied based on the disk capacity usage ratio s_{disk} and incoming workload properties. Generally, there are two kinds of GC processes: Background GC process, which cleans the dirty data only during the disk idle period, and it has a lower

priority compared with normal user requests; Foreground GC process, which handles the dirty data when $s_{disk} > 0.5$, and the disk does not have enough number of idle periods. The foreground GC has the same priority as the normal user requests.

4.3.1 Background GC Process

With background GC process, the GC is only performed when the disk is in idle mode. The GC process being processed can be canceled when there are user requests arriving at the disk internal queue. Thus the overall performance degradation caused by the background GC process can be neglected and we can simplify (4.1) as:

$$\min \quad \beta P_c - \alpha C_s, \quad (4.2)$$

where $C_s = b_2 S_{req} r_{dirty}^t$ is the dirty data being cleaned in one GC process, and $P_c = P_c^{read} + P_c^{write}$ is the energy consumption by one GC process. The P_c^{read} and P_c^{write} can be calculated through (4.3).

$$\begin{aligned} P_c^{read} &= (P_{seek}^r + P_{rot}^r) \times b_2 \times r_c + P_{tran}^r, \\ P_c^{write} &= (P_{seek}^w + P_{rot}^w) / (1 - r_c) + P_{tran}^w, \end{aligned} \quad (4.3)$$

where $P_{tran}^r = \gamma_r S_{req} b_2 (1 - r_{dirty})$ and $P_{tran}^w = \gamma_w S_{req} b_2 (1 - r_{dirty})$, γ_r is the energy consumption for transferring 1 read block, and γ_w is the energy consumption for recording 1 data block.

By solving the equations, we can get the optimal r_{dirty}^t as shown in (4.4)

$$r_{dirty}^t = \frac{A + B}{b_2 S_{req} + B}, \quad (4.4)$$

where $A = \frac{\beta}{\alpha}(b_2(r_c P_{seek}^r + P_{rot}^r) + (P_{seek}^w + P_{rot}^w)/(1 - r_c))$, $B = \frac{\beta}{\alpha}(\gamma_r b_2 S_{req} + \gamma_w b_2 S_{req})$.

From the results, we can see that the r_{dirty}^t increases when the b_2 or S_{req} decreases.

4.3.2 Foreground GC Process

By applying foreground GC process, as the idle time is not enough for the GC process, and $s_{disk} > 0.5$. Foreground GC might be required to help cleaning the wasted space caused by the invalid data. The overall drive performance degrades as the foreground GC consumes a certain percentage of the disk processing power. The performance degradation cannot be eliminated from (4.1).

The importance of the capacity saving C_s through GC process is dependent on the remaining capacity of the disk. We can adjust the weight factor α based on the available disk capacity. If the disk capacity is still large enough, the optimization problem can be formulated as (4.1), where the α is adjusted based on the status of the disk.

To solve the problem formulated, we need the following steps: Firstly, we need to find out the disk capacity cleaned by the GC process, which is $C_s = b_2 S_{req} r_{dirty}^t$; Then, the energy consumption of the GC process can be calculated through (4.5); Lastly, the performance degradation can be estimated by considering the percentage of disk bandwidth allocated to the GC process, which is based on the disk fullness status. As this chapter mainly focuses on the cost saving on the energy consumption and disk capacity, we can fix the performance degradation caused by the foreground GC process, which can be denoted as $P_d = \beta_1(\lambda_{max} - \lambda_{max}^{gc})$.

$$\begin{aligned}
P_c &= P_c^r + P_c^w \\
&= P_{sk}^r + P_{rot}^r + P_{tran}^r + P_{sk}^w + P_{rot}^w + P_{tran}^w, \\
P_{tran}^r &= \gamma_r S_{req} b_2 (1 - r_{dirty}), \\
P_{tran}^w &= \gamma_w S_{req} b_2 (1 - r_{dirty}).
\end{aligned} \tag{4.5}$$

Then the problem (4.5) can be solved to get the optimal GC dirty ratio threshold r_{dirty}^t , which is shown in (4.6).

$$r_{dirty}^t = \frac{A + B + P_d}{b_2 S_{req} + B} \tag{4.6}$$

where $A = \frac{\beta}{\alpha} (b_2 (r_c P_{seek}^r + P_{rot}^r) + (P_{seek}^w + P_{rot}^w) / (1 - r_c))$, $B = \frac{\beta}{\alpha} (\gamma_r b_2 S_{req} + \gamma_w b_2 S_{req})$.

Based on (4.6), it is clear that when the performance degradation increases, r_{dirty} will also increase. Meanwhile, when b_2 or S_{req} increases, we can set smaller r_{dirty} to increase the overall cost saving.

4.4 Results and Analysis

In this section, the r_{dirty}^t selection methods are evaluated based on the SMR drive simulator shown in Chapter 3. To study the energy consumption, we create a power model in the disk simulator module to estimate the energy consumption, including the energy usage for seeking, rotation and data transfer. Meanwhile, to study the disk space saved, the number of dirty data blocks cleaned by the GC process is also recorded.

Based on the simulator, the overall results under different workload properties are

Figure 4.2: Cost savings under different request sizes and r_{dirty}^t

shown in 4.4.1. Meanwhile, the results under different system settings are shown in 4.4.2. The cost savings under different workload properties and drive settings are presented with different price ratios, which are the cost of 1KWH energy over the price of 1GB disk capacity. Due to the growing technologies of the HDD drive, the price of disk capacity will keep decreasing, and the price of energy consumption will keep increasing. Thus different cost ratios are utilized to present the requirements for current and future situations.

4.4.1 Results under Different Workload Properties

4.4.1.1 Request Size

The request size affects the system performance, e.g. IOPS and response time, as the data transfer time is decided by the size of data processed. Besides the system performance, the disk energy consumption and the capacity cleaned in each GC process are also affected by the request size. With constant GC batch size, the larger the request size, the more data will be processed by the GC process in each time, and the energy consumed by GC process also increases. The cost savings of GC process under different request sizes are shown in Fig. 4.2. From the figure, we

can observe that the benefit of GC process is larger when the request size is bigger and r_{dirty}^t is larger. This is on the account of that the size of space cleaned with a GC process increases if the request size and r_{dirty}^t are larger, while the energy consumption only increases a bit for the data transfer process.

4.4.1.2 Read Ratio

The cost savings of GC process under different read ratios are shown in Fig. 4.3 and Fig. 4.4. From the figures, we can notice that the optimal r_{dirty}^t is not affected by the read ratio. This is because the selection of r_{dirty}^t is based on the relationship between the energy consumption and the disk space cleaned. Both of these factors are not affected by the number of GC processed. Along these lines, from the figures, we can also notice that the benefit of GC process increases faster when the read ratio is smaller. This is on the account of that there are more write requests, which can be processed in a batch to save the disk bandwidth, and the disk can have more idle time to handle the GC processes. From the figures, we can also find out that when the ratio of energy consumption cost over the cost of disk capacity saving increases, the optimal r_{dirty}^t also increases for all the cases. The principle explanation behind this is that the cost of energy consumption for the GC process increases more compared with the benefit of capacity saving.

4.4.2 Results under Different System Settings

In this section, the cost savings with GC process under different system settings are evaluated. The system settings include s_{disk} , b_2 , t_{idle}^t and b_1 , etc.

4.3.1: ratio=25

4.3.2: ratio=100

Figure 4.3: Cost savings under different read ratios and r_{dirty}^t . Figure 4.4: Cost savings under different read ratios and r_{dirty}^t with cost ratio 50.

4.5.1: ratio=25

4.5.2: ratio=100

Figure 4.5: Cost savings under different disk statuses and r_{dirty}^t .

4.4.2.1 Disk Status

The cost savings under different disk statuses are illustrated in Fig. 4.5 and Fig. 4.6. In the figures, when the price ratio is **25**, the most suitable r_{dirty}^t is around **0.1** when the disk space is nearly full, and around **0.2** when the disk space is around **50%** full. The cost saving increases faster with the increased r_{dirty}^t , and the decreasing speed of cost saving is also faster with increasing s_{disk} . This can be explained by the fact that the disk bandwidth allocated for GC processes increases when the s_{disk} is larger, as the number of GC processes increases, and the cost saving increases as well. With the increased price ratio, the r_{dirty}^t increases as well. An explanation behind this is that when the cost of energy consumption increases, the cost of GC process increases as well, while the benefit of capacity cleaning does not change.

4.4.2.2 GC Batch Size

Fig. 4.7 and Fig. 4.8 show the cost savings of GC process under different GC batch sizes and r_{dirty}^t . When the price ratio is **100**, the optimal r_{dirty}^t increases from **0.3** to **0.9** when b_2 reduces from **32** to **2**. This is because the larger GC batch size linearly increases the number of dirty data cleaned while the energy

Figure 4.6: Cost savings under different disk statuses and r_{dirty}^t with cost ratio 50.

4.7.1: ratio=25

4.7.2: ratio=100

Figure 4.7: Cost savings under different GC batch sizes and r_{dirty}^t .

consumption only slightly increases by the data transfer process. Moreover, the cost saving increases faster with the increased r_{dirty}^t , as more invalid data can be cleaned with the same amount of energy consumed. When the price ratio reduces, the optimal r_{dirty}^t decreases, as the cost of energy consumption reduces more compared with disk capacity cleaned.

4.4.2.3 Idle Time Threshold

The cost savings under different t_{idle}^t and r_{dirty}^t are shown in Fig. 4.9 and Fig. 4.10. Under the 3 different price ratio settings, the optimal r_{dirty}^t is the same for

Figure 4.8: Cost savings under different batch sizes and r_{dirty}^t with cost ratio 50.

4.9.1: ratio=25

4.9.2: ratio=100

Figure 4.9: Cost savings under different idle time thresholds and r_{dirty}^t .

Figure 4.10: Cost savings under different idle time thresholds and r_{dirty}^t with cost ratio 50.

4.11.1: ratio=25

4.11.2: ratio=100

Figure 4.11: Cost savings under different host write batch sizes and r_{dirty}^t . Figure 4.12: Cost savings under different host write batch sizes and r_{dirty}^t with cost ratio 50.

all the idle time threshold settings, while the cost saving increases faster if the idle time threshold is smaller. This can be explained by the fact that t_{idle}^t only affects the number of GC processes operated, but the cost of each GC process remains the same.

4.4.2.4 Host Write Batch Size

The cost savings under different host write batch sizes and r_{dirty}^t are shown in Fig. 4.11 and Fig. 4.12. The figures present that the cost saving increases when b_1 is larger and r_{dirty}^t increases. This is because of that when the write batch size increases, more write requests will be grouped together, and the number of write requests to the disk reduces, which means more disk idle time can be utilized for the GC processes.

4.5 Conclusion

In this chapter, an analytical model is presented to explain the effectiveness of r_{dirty}^t selection. In the model, the cost of energy consumption and the benefit of disk capacity cleaned with the GC process are considered, and the optimal r_{dirty}^t can be calculated by solving an optimization problem. Based on the model and the simulator introduced in Chapter 3, we simulate the performance of SMR drive, especially the energy consumption and the number of dirty data blocks cleaned. The simulation results show reasonable cost saving trends under different r_{dirty}^t , drive parameters, and workload environments. Based on the results, the r_{dirty}^t can be tuned to have maximum benefit by balancing the cost of energy consumption and the saved disk capacity.

After studying the performance and energy consumption of the SMR drive, which

is the backup tier in the data center storage system, we can move to the higher level of storage architecture. In the next chapter, the performances of different types of hybrid storage systems are analyzed and compared, including the overall system response time, energy consumption and SSD lifespan utilization.

Chapter 5

Hybrid Storage System Simulation and Comparison

5.1 Background

After we have the detailed analysis of the performance and power consumption of storage devices, especially the SMR drive in a storage system, it is important to understand the performance of a storage system, which consists of different storage devices. As shown in Chapter 1, the hybrid storage system is still the main-stream in current data center design, especially those hybrid storage systems comprise HDDs and SSDs. It will continue to be the main-stream for future data center storage systems, if the performance and cost differences of various types of storage devices still exist.

In Chapter 2, we described various types of hybrid storage architectures, including the SSD tiering method, SSD caching method, SSD hybrid method and a few special architectures, such as HDDs as the cache to SSDs. Meanwhile, various algorithms

and policies utilized in the hybrid storage system are also introduced, such as the hot data identification, data migration, address mapping, data allocation, and request scheduling, etc.

With such a huge number of storage system architectures and algorithms, it is important to know the performance of these systems under various workloads. In this chapter, a general storage system simulator is designed and developed to support for the simulation and comparisons of these basic storage architectures, such as SSD tiering method, SSD caching method, and SSD hybrid method. Meanwhile, different cache policies are also applied, such as the read-only policy, and write-back policy.

The main contributions of this chapter are as follows:

- A storage system simulator is designed and developed to support different kinds of storage architectures, such as SSD tiering method, SSD caching method, and SSD hybrid method. Different algorithms and policies are also supported in the simulator.
- The performances of various storage systems under different workload environments are analyzed and compared. The impacts of different cache policies and algorithms on the systems are also studied.

The rest of this chapter is organized as follows. Section 5.2 presents the simulation tool, and the algorithms utilized. Section 5.3 provides the hardware specification and system settings. Section 5.4 compare and discuss the performances, including the overall response time, SSD lifespan usage, and the energy consumption thoroughly for different caching policies. Section 5.5 compare the performance under various types of system architectures. Finally, Section 5.6 draws the conclusion.

Figure 5.1: The overall structure of the hybrid storage system simulator

5.2 Simulation Design and Setup

In history, the hybrid storage system simulation is studied and analyzed by numerous research works. OGSSim [161] is one of the storage simulators to study the performance of different storage systems with generic devices and architecture layouts. [162] compares the performance of different storage systems, such as tiered storage, distributed storage and normal HDD storage. StorageSim [163] is developed to simulate the performance of a multiple-tiered storage system under different data placement policies. However, all these simulation tools only focus on one or two aspects of the hybrid storage systems, and lack of comparisons between the basic architectures, such as SSD caching method and SSD tiering method. To study the performance, such as response time, energy consumption and SSD lifespan, under different kinds of hybrid storage systems, a hybrid storage simulator is designed and developed.

Our hybrid storage simulator contains three components, the HDD simulator, the SSD simulator and the hybrid controller, which is shown in Fig. 5.1. The hybrid controller is implemented differently under different storage architectures. The architecture with SSD tiering method contains a tier handler, which handles the

data migration and request re-direction, a hot data identification module, which detects the hot data. The architecture with SSD caching method comprises a cache handler, which has an LRU cache policy implemented. The architecture with SSD hybrid method consists of a tier handler, a cache handler and a hot data identification module. The data movement between the HDD and SSD inside the cache and tier components are processed by the tier and cache handler. In the cache handler, there are two types of caching policies implemented: read-only policy and write-back policy.

It is difficult to verify the hybrid storage system performance with actual system or prototype, as there are a lot of unknown components inside the system, which cannot be fully covered by the simulation. However, we can make sure the correctness of simulation for the storage system components, such as the HDD/SSD devices, and data link performance, etc. We have conducted a validation by measuring the actual device performance and compared the measured results with our simulation results. They show quite similar performance. Meanwhile Omnetpp [164] is utilized to simulate the network links and data links.

5.3 System Specification

In the simulator, the HDD device specification is shown in Table 5.1, and the specification of SSD device is shown in Table 5.2. The traces utilized in the simulation are listed in Table 5.3. There are **3** types of traces utilized in the simulation: 1). SPC1C trace [165] is an industry-standard storage performance benchmark, and it represents the end user applications. 2) Financial trace [166] is the workload gathered through running the OLTP applications at two large financial institutions. 3). Web searching trace [166] is gathered from a popular search engine. The trace property re-access ratio is the number of requests that access the previous existing data

Table 5.1: The hardware specification of HDD in the hybrid storage system

Property	Value
HDD capacity	600GB
HDD RPM	7200
HDD maximum seek time	11ms
HDD number of tracks	Around 300K
HDD sectors per track	2048 blocks average
HDD idle power	3.7W
HDD busy power	5.7W
HDD DRAM size	64MB
HDD DRAM cache line size	256 blocks
HDD cache policy	LRU

Table 5.2: The hardware specification of SSD in the hybrid storage system

Property	Value
SSD read time	0.2ms
SSD write time	0.268ms
SSD erase time	1ms
SSD read power	2W
SSD write power	2W
SSD idle power	0.05W
SSD page size	256 blocks
SSD block size	256 pages
SSD cache policy	LRU

Table 5.3: The different types of workloads and their properties

Workload	Write ratio	Reaccess ratio	IOPS	Length	Access frequency
SPC1C trace	60%	15.71%	25-325	12K-155K	1.5733
Financial trace	78%	99.4%	336.6	4.413M	45.31
Web searching trace	0.1%	60.2%	334.78	1.055M	4.7031

over the total number of requests. The access frequency is the average number of accesses for each of the accessed range (the range size is set to be same as the SSD page, which is 64K), which is an important factor to check the hotness of the data. Moreover, the hot data identification threshold is also set differently for different traces, e.g. **2** for SPC1C trace, **3** for web searching trace, and **5** for financial trace. All these parameters can be adjusted through the system settings.

5.4 Performance Comparisons under Different Caching Policies

In this section, the performances of different hybrid storage architectures are compared and analyzed. A hybrid storage system simulator is designed and developed to conduct a quantitative comparison of different storage architectures, e.g. SSD as a read-only cache, SSD as a write back cache, and SSD as a high tier, etc. The comparisons are conducted based on the following perspectives: a). The system performance, such as the overall response time. b). The SSD lifespan, which is simulated as the number of data written and erased in the SSD device. c). The energy consumption, which includes energy consumption caused by HDD and SSD. To make the comparison more meaningful and intuitive, we focus on the comparisons between the different SSD caching architectures and the SSD caching and SSD tiering policies in their basic forms, which are under the same hardware settings and workload environments. Note that although our settings are straightforward, the results can also reflect the insight of the complex systems with disk arrays, as the corresponding array can be virtualized as a storage pool.

Table 5.4: Performance comparison under different caching policies for financial trace

Caching mode	Performance(s)		SSD lifespan		Energy(KWh)	
SSD free space	Limited	Enough	Limited	Enough	Limited	Enough
Read only	0.000300	0.000201	1294900	1496875	0.014494	0.014262
Write back	0.000687	0.000616	4157523	4113167	0.015396	0.014577

Table 5.5: Performance comparison under different caching policies for web searching trace

Caching mode	Performance(s)		SSD lifespan		Energy(KWh)	
SSD free space	Limited	Enough	Limited	Enough	Limited	Enough
Read only	0.008755	0.000930	553215	77396	0.004682	0.003590
Write back	0.008755	0.000930	553439	77527	0.004683	0.003590

5.2.1: Enough SSD free space

5.2.2: Limited SSD free space

Figure 5.2: System performance under different caching policies for SPC1C traces.

5.4.1 Performance

When the free space of SSD is enough, the caching architecture with write-back policy has better performance than the read-only policy for the workloads SPC1C and OLTP, which contains a certain amount of write requests. The performance comparisons for the SPC1C traces are shown in Fig. 5.2.1, and the performance comparisons for OLTP traces are shown in Table 5.4. The performance is similar for the read-intensive workloads, such as web searching trace, which is shown in Table 5.5. The main reason for this is that the write requests are cached in the SSD device for the architecture with write-back policy, which lowers the response time of the write requests. Meanwhile, the number of requests to the HDD device reduces and the performance of read requests to HDD improves as the load of HDD reduces when the IOPS of the workload is large. As the SSD space is enough to cache the write data and hot read data, the flushing of write data to HDD only needs to be handled during the system idle time, the overhead of write data flushing is minimized.

As shown in Fig. 5.2.2, when the free space of SSD is not sufficient, the overall response time increases for both the read-only policy and write-back policy for the workloads with high IOPS. The increment for the write-back policy is significantly larger compared with read-only policy, as the write requests to SSD might cause write data flushing to HDD when the free space in SSD is not enough to store the new coming data, and the GC process inside the SSD device. The performance of the system with read-only policy also decreases, which is caused by the reduced number of hot data kept in the SSD cache, which might reduce the chance of read cache hit. Meanwhile, we also notice that there is a turn point around BSU 40, where 1 BSU equals to 5 IOPS. When BSU is small, the number of requests to HDD is small, there are enough idle periods for the cache to flush the write data to HDD, and the DRAM cache has clean space for storing the new coming write

requests. Thus the read-only cache has better performance compared with write-back cache. However, when the BSU is large, the number of idle periods may not be enough to let the DRAM cache flush the write data, and the process of write data flush will increase the waiting time for the requests. In this circumstance, the write-back policy can keep the write data for a while, and have better performance compared with read-only cache policy.

In Table 5.4, the performances of systems with both caching policies for the financial trace are compared. We can notice that the performance with read-only policy is somewhat superior to the one with write-back approach, in light of the very high re-access ratio of the workload. Since the majority of the requests can be dealt by the cache in SSD and DRAM, the disk has enough idle periods to handle the flushed write data, which cleans the dirty data inside the DRAM cache. The cleaned space can be utilized to handle the future coming write requests. Thus the performance of hybrid storage system with SSD as the read-only cache is slightly better compared with write-back policy, on the grounds that the write access speed of DRAM is faster than SSD. In Table 5.5, the performances of the systems with both cache policies for web searching trace are compared, and we can see that the performance is almost the same. This is on account of that the web searching trace is a read-intensive workload, which has very few write requests.

To reduce the overhead of data flushing and data copying between HDD and SSD, [38] identifies the hot data at the host side by analyzing the user level information, which provides more hints for the identification process. With the accurate hot data identification process, the system can minimize the cold data copying to SSD, and increase the chance of read cache hit. In [44], the lazy adaptive cache replacement algorithm can also reduce the chance of copying cold data to SSD, which also improves the read cache hit ratio and the overall system performance. In [42], the demotion operations are counted as the reference to decide the data movement between SSD

5.3.1: Enough SSD free space

5.3.2: Limited SSD free space

Figure 5.3: SSD lifespan under different caching policies for SPC1C traces.

and HDD, which has similar effects as setting the data hotness count utilized during the hot data identification algorithm. To improve the data replacement policy for the write cache, [46, 47] split the cache space into read and write regions, and read region only handles the hot read data, while write region handles the write requests. This caching policy improves the cache replacement performance and enhance the efficiency of the GC process, which improves the overall system performance if the workload is similar to the SPC1C trace, which has near number of read and write requests. However, if the workloads are similar to the OLTP trace, which are read intensive or write intensive, the corresponding write cache region and read cache region will be wasted and result in performance degradation.

5.4.2 SSD Lifespan

In Fig. 5.3, the sizes of write data to the SSD, which affect the SSD lifespan most under different caching policies for the SPC1C traces are presented. We notice that the system with write-back cache policy writes more data to the SSD, this is the result of that the write requests are taken care of by the SSD devices. Meanwhile, the read data are likewise duplicated to the SSD cache from HDD for future read

access. From the figure, we also notice that the size of write data to SSD is larger when the SSD free space is not enough to cover the incoming write data, this is caused by the GC process in SSD, which consists of the processes of reading data, erasing block, and writing data.

In Table 5.4, the number of SSD writes in the system with read-only policy is significantly less compared to the one with write-back policy. This is on the ground that the write requests are mostly handled by the HDD device. The numbers of SSD writes for systems with read-only and write-back policies are similar for the web searching trace as shown in Table 5.5, which can be clarified by the high read ratio of the workload.

Minimizing the number of writes to SSD [48,52] can prolong the SSD lifespan at the cost of decreasing the overall system performance when the capacity of the SSD is not fully utilized. In [38], the SSD device only caches for the random write data, and the sequential write data are located to the HDD, which prolongs the SSD lifespan, which is suitable for the OLTP traces. Meanwhile, host side information is utilized to check the hotness of new coming data, which can provide more accurate prediction and thus minimize the chance of cold data migration to the SSD devices, and thus prolongs the SSD lifespan. In [39], before the write request coming to the storage system, deduplication process is conducted to remove the duplicated data, which can reduce the number of updates significantly when the workload contains lots of update requests. At that point, the SSD lifespan can be prolonged. In [44], due to the lazy adaptive cache replacement algorithm, the insertion and removing of cache entries from the SSD are more strict, which reduces the number of write operations to SSD and prolongs the SSD lifespan. Meanwhile, the GC process required cleaning the SSD space and the data flushing requests can also be reduced, which ensures the system has a consistent performance. However, it reduces the overall system performance when the SSD space is not fully utilized.

5.4.1: Enough SSD free space

5.4.2: Limited SSD free space

Figure 5.4: System energy consumption under different caching policies for SPC1C traces.

5.4.3 Energy Consumption

The energy consumptions of the system with various caching policies for SPC1C traces are shown in Fig. 5.4. When SSD has enough free space, the system with read-only cache policy consumes more energy compared with the one applying write-back policy. This is due to that the write requests are mostly handled by HDD device for the system with read-only cache policy, which increases the HDD busy time and energy consumption. When SSD only has limited free space, the energy consumption is higher for the system with the read-only cache policy when the IOPS is low, but lower when the IOPS is high compared with the system with write-back policy. An explanation behind this is that when the free space in SSD is not enough, if the IOPS of the trace is higher, the busy period of the HDD device is similar for both read-only cache policy and write-back policy, as the write data needs to be flushed to HDD under write-back policy. Meanwhile, the SSD device consumes more energy under write-back policy as it needs to handle more write requests. In Tables 5.4 and 5.5, we can observe that lower energy consumption of the system with read-only policy when the SSD free space is limited. This is because the write requests need

to be handled by SSD device first, and then flushed to HDD in the system with write-back policy, which consumes more energy in SSD. When the free space of SSD is enough, the energy consumption is similar for both cache policies as the write data do not need to be flushed to HDD since the cache space is enough to cover the incoming write data.

Minimizing the number of writes to SSD [44, 48, 52] can prolong the SSD lifespan at the cost of higher energy consumption as more write requests are handled by HDD devices when the SSD space is not fully utilized. However, fewer data flushing and GC processes need to be handled, which saves the energy consumption. Deduplication process utilized in [39] saves the overall energy consumption as less write requests need to be handled by SSD and HDD when the workload contains a lot of duplicated access data, such as OLTP workload. Host side information is utilized to check the hotness of new coming data in [38], which can also save the overall energy consumption of the storage system as the unnecessary data migrations between HDD and SSD are minimized.

5.5 Performance Comparisons under Different System Architectures

In this section, we will discuss the performances of different storage architectures under different kinds of workloads. In order to make sure that the comparisons are fair, all the architectures apply read-only cache or tier policy in their system, where the SSD devices are only utilized to store the hot read data. The performance comparisons for the systems handling financial and web searching traces are listed in Tables 5.6 and 5.7, and the performances for SPC1C traces are shown in Fig. 5.5, 5.7 and 5.9.

Table 5.6: Performance comparison under different system architectures for financial trace

Caching mode	Performance(s)		SSD lifespan		Energy(KWh)	
SSD free space	Limited	Enough	Limited	Enough	Limited	Enough
SSD Cache	0.000300	0.000201	1294900	1496875	0.014494	0.014262
SSD Tier	0.000262	0.000198	1180351	1326725	0.014408	0.014245
SSD Hybrid	0.000219	0.000201	1282203	1496864	0.014358	0.014262

Table 5.7: Performance comparison under different system architectures for web searching trace

Architecture	Performance(s)		SSD lifespan		Energy(KWh)	
SSD free space	Limited	Enough	Limited	Enough	Limited	Enough
SSD cache	0.008755	0.000930	553215	77396	0.004682	0.003590
SSD tier	0.020085	0.009704	78340	46041	0.004580	0.004137
SSD hybrid	0.011454	0.000930	559622	77396	0.004727	0.003590

5.5.1 Performance

In Fig. 5.5, we can notice that the performance of the system with caching method is better than the one with tiering method and hybrid method, regardless of whether the SSD free space is sufficient or not. This can be clarified by the properties of SPC1C traces. Due to the small re-access ratio of SPC1C traces, the number of hot data migrated to SSD is less compared with SSD caching method and the benefit of faster SSD read access can not be fully utilized with the tiering method as the read cache hit ratio reduces. We additionally see that the system with the hybrid method has slightly better performance than the tiering method when the SSD free space is not enough. This is in light of the fact that the combination of tiering and caching method can fully utilize the benefit of caching method when the re-access ratio of the workload is smaller. Comparative results can be observed for the web searching trace, which also does not have high re-access ratio. However, when SSD free space is limited, if the re-access ratio of the workload is high, such as the financial trace, the performance of tiering method is better compared with caching method, which can be found in Table 5.6. This is because the tiering method minimizes the data

5.5.1: Enough SSD free space

5.5.2: Limited SSD free space

Figure 5.5: System performances under different storage architectures for SPC1C traces.

migration between the SSD and HDD and only the hot data are migrated to the SSD. To further improve the system performance, [102] considers the load status of the SSD and HDD devices during the data allocation and migration, which improves the system performance by parallel usage of the devices.

In a tiered storage system, the system performance is also highly affected by the hot data identification and data migration policies, which can be observed in Fig. 5.6. From the figure, it is easy to notice that the system performance decreases when the data hotness threshold increases. This can be explained by the fact that the size of data migrated to SSD reduces when the hotness threshold increases. The benefit of the faster accessing speed of SSD can not be fully utilized compared the one with lower hotness threshold. More read requests need to access data from HDD, which lower the overall system performance.

To improve the tiered storage system performance, Hystor [78] tries to save the critical access data, which are found out by checking the access locality, to SSD tier. Meanwhile, the write data is also allocated to SSD, which improves the write performance and the overall performance when the SSD device is not fully written. However, when the SSD space is nearly used up, the performance will degrade sig-

Figure 5.6: The system performances under different hot data identification policies for SPC1C traces.

nificantly, as GC operations are required to clean the SSD space for future usage, which consumes lots of disk processing bandwidth. In [95], together with the hotness of the data, the other data properties, such as randomness and data size, are also considered during the data migration, which can further improve the system performance, as it fully utilizes the performance properties of the SSD and HDD devices.

5.5.2 SSD Lifespan

The SSD lifespan usages under various storage architectures for SPC1C traces are shown in Fig. 5.7. From the figure, we notice that the SSD lifespan usage under the tiering method is vastly improved compared with the caching method and hybrid method. This is because of the fact that the tiering method only moves the data to SSD when the data is identified as hot, while the caching method moves the data to SSD when it is read from HDD. In Tables 5.6 and 5.7, the SSD lifespan usages with financial trace and web searching trace are shown. We notice that the SSD lifespan usages under the tiering method are always smaller compared with the caching method and the hybrid method for both of the traces no matter the SSD

5.7.1: Enough SSD free spacey

5.7.2: Limited SSD free space

Figure 5.7: SSD lifespan usage under different storage architectures for SPC1C traces.

free space is enough or not. In order to improve the overall system performance, the load balance is also considered. In [102], the SSD lifespan is further utilized as the proposed algorithm needs to maximize the parallelism of the devices, which requires more write operations to SSD when the load of HDD is very high.

In a tiered storage system, the SSD lifespan is also affected by the hot data identification policies, which can be shown in Fig. 5.8. From the figure, we notice that when the hotness threshold is large, the more SSD lifespan prolongs. This can be explained by the fact that the size of hot data moved to SSD reduces when the hotness threshold is large compared with the one having a smaller hotness threshold. Hybrid store [80] utilizes the historical access frequency information to distinguish the hot data, which is similar as the data hotness threshold utilized in our simulated system.

5.5.3 Energy Consumption

In Fig. 5.9, the energy consumptions of the systems with various architectures are shown. From the figure, we can observe that the tiering method consumes more

Figure 5.8: The SSD lifespan usage under different hot data identification policies for SPC1C traces.

5.9.1: Enough SSD free space

5.9.2: Limited SSD free space

Figure 5.9: System energy consumption under different storage architectures for SPC1C traces.

energy when the IOPS of the workload is smaller no matter whether the SSD free space is enough or not. This is on the ground of that the read request needs to access data from HDD several times before the data is migrated to SSD, while the data is directly copied to SSD for the first access in the system with caching method or hybrid method. When the IOPS of the workload is large, the load of the HDD is mostly full, and the difference between the energy consumptions of HDD for the tiering method and the other two architectures is smaller. Meanwhile, the tiering method has less SSD accesses and the energy consumption is less compared with the other two architectures. Along these lines, the overall energy consumptions of the three architectures are comparative when IOPS of the SPC1C trace is high. The energy consumptions of the different architectures for web search and financial traces are shown in Tables 5.6 and 5.7. From the tables, we can notice that the energy consumption of the system with tiering method is slightly smaller compared with caching and hybrid method when the SSD capacity is enough. This can be explained with the properties of these two workloads. The financial trace has a very high re-access ratio, and the HDD is mostly in idle, the read accesses to HDD are mostly handled by the SSD cache or the DRAM cache in HDD. Meanwhile, the higher number of data copying operations between HDD and SSD in the system with caching method may consume more energy. As the IOPS of web searching trace is high and re-access ratio is small, the HDD is busy for most of the time, thus the energy consumption of HDD is similar for all the three architectures. However due to the fewer data migrated to SSD in tiering method compared with the other two architectures, the energy consumption of SSD is much less. Therefore the overall energy consumption is less for tiering method.

5.6 Conclusion

In this chapter, we present a hybrid storage system simulator that supports for different types of architectures and algorithms. Based on the simulator, the advantages and disadvantages of some basic hybrid storage architectures are studied and compared quantitatively in several aspects, such as the system performance, SSD lifespan, and energy consumption, etc. Although it is generally difficult to provide quantitatively comparative results for all the systems due to the system complexity with a large number of different algorithms in diversely hybrid structures, the simulated comparative results can still act as a basic benchmark to analyze the performance of different systems.

Research works on integration of the tiering method and caching method to improve the performance of the storage system are promising. However, the current integration techniques are only done at a basic combination level, and the impact of the integration is not analyzed in detail. At the same time, artificial intelligence and machine learning techniques have shown the capabilities in many industrial applications. One possible future direction is to investigate the integrated platform with those techniques for a better adaptivity to dynamic workload and wider applications to various scenarios with tighter performance, reliability, cost, lifespan and energy requirements.

Chapter 6

New Hybrid Disk Array Structures with Consistent Performance and Prolonged Life

6.1 Introduction

As shown in the previous chapters, hybrid storage systems are deployed by industry players and researchers to address both performance and capacity concerns with minimal cost, which fully take the advantages of high capacity of low tier devices and high performance of high tier devices. Currently, the most popular hybrid storage system still uses SSD as the faster tier device and HDD as the low tier device.

In a hybrid storage system, SSD and HDD are usually utilized separately, i.e., pure SSDs or pure HDDs are formed as a storage pool. These storage pools would then be mixed to form various types of hybrid structures. The fast speed SSD devices or the

SSD pool can be either placed as the cache or high-performance-tier. This is due to that SSD has much higher performance in random access than HDD, i.e., hundreds even thousands times faster. However, when considering the sequential access, HDD can still compete with SSD in specific cases, such as in the level of **200Mbps** throughput. Specifically, the SMR drives are in favor of sequential accesses. They combine the random write requests together and write them to disk sequentially, which further enhances the random write performance. Other than that, SSD has a limited number of write cycles, such as an MLC NAND may only have less than **100k** write cycles. A lot of write operations to the SSD will consume these write cycles fast and decrease the SSD lifespan.

As is known, HDD has a similar level of sequential performance compared with SSD, and HDD has a much lower cost compared with SSD. To spare the total cost while keeping up the comparative performance, a natural question is whether HDDs and SSDs can be mixed in the same pool or disk array if the IO pattern of incoming workload is the sequential-access dominated, such as the workload from video surveillance system [167].

To take care of the issue of limited SSD write cycles, researchers have proposed lots of storage algorithms [39, 44, 53, 74] and architectures [48] to avoid the unnecessary writes to SSD. Some of them [42, 43, 53, 66] apply SSD as the read-only cache to save the SSD lifetime, however, it decreases the write performance. Some researches [107, 108] even treat HDD as the write cache for SSD to save the SSD lifetime at the cost of reducing overall system performance. In this way the second question is why not put write data on HDD in the hybrid storage system whenever possible.

In the conventional hybrid storage design, SSD is generally the place to hold the hot data for either cache or tier. The hot data for the most part refer to the data that read frequently in a given time window. However, for write access, it also has hot data in the similar sense. The most effective method to deal with read hot data

(RHD) and write hot data (WHD) in the same tier of hybrid storage system turns into another common question. Here, the RHD and WHD are the frequently and/or recently accessed read hit on read (ROR) data and write hit on write (WOW) data, respectively.

To tackle the inquiries previously mentioned, in this chapter, we present a new hybrid structure configuration method. In the strategy, the HDDs are formed as stripes in RAID0 design when necessary to ensure that its sequential performance is near to SSDs. These HDD sub-array and some SSDs can additionally form a hybrid sub-array. Meanwhile, a new data allocation method as shown in Fig. 6.1 is proposed to allocate all RHD to SSD and all WHD to HDD with the guide of a hot data identifier. An appropriate decision algorithm is proposed to figure out which device the data needs to move if the data is hot in aspects of read and write. In spite of the fact that this chapter concentrates on the hybrid storage array in view of the SSDs and HDDs, the proposed solution can likewise be applied to different types of devices, such as PCM, PRAM, and so forth, which have performance and price difference.

The main contributions of this chapter are as follows:

- A new hybrid structure configuration method is proposed to arrange the HDDs and SSDs in a hybrid storage system, which saves the total cost of the system, while maintaining the system performance.
- A new data allocation method is proposed to allocate the data in view of the data properties, such as hotness, read/write, and sequential/random. The SSD lifespan prolongs while the overall system performance maintains.
- The data allocation method additionally deals with the device parallelism and the disk statuses, such as remaining free disk spaces. The system parameters are adjusted based on the disk statuses to further enhance the overall system

performance.

The remainder of this chapter is organized as follows. Section 6.2 discusses the historical research works related to the hybrid storage system, especially the hybrid array. Section 6.3 presents the system configuration and data allocation methods. Section 6.4 analyzes the system performance based on the IOPS, throughput and SSD lifespan saving. Section 6.5 shows some simulation results. Section 6.6 discusses about the simulation results. Finally, Section 6.7 draws the conclusion.

6.2 Background

Hybrid storage systems that contain HDD as the capacity tier and SSD as the cache or performance tier are quite popular in the research area.

To fully utilize the fast random access property of SSD, some hybrid storage systems use SSD as the cache layer between DRAM and HDD. The hot read or the write data are cached in SSD to improve the accessing speed and overall system performance. There are mainly two types of architectures in this category: SSD is used as read-only cache [42] [43] [53] [66] and SSD is used as both read and write cache [67] [63] [68] [69] [70].

In tiered structure, SSD is usually utilized to store the performance critical data and HDD is used to store the cold data. Based on the location of the controller, the tiering method can be categorized as host controlled [79] [81] [82] [85] [88] [93] [94] and device controlled [78] [83] [84] [86] [87] [89].

In host controlled tiered structure, the decision of which tier to store the incoming data is decided by the host through analyzing the host side information. A linearly weighted formulation [82] is used at the operating system level to characterize the workload access pattern to enhance the overall I/O performance. A versatile tiering

hybrid drive called GreenDM [79] combines the Flash-based SSDs and HDDs in one storage system, and a block level interface is provided to allow data migration between faster SSD and slower HDD. The GreenDM also provides tunable parameters to adapt the system to different kinds of workloads.

In device controlled tiered structure, the decision of data movement is made by the drive controller through analyzing the historical accessing information and the properties of the incoming data. Hystor [78] analyzes the data access pattern, especially the access locality to find the critical accessing data. Hystor saves the critical data to the SSD tier to improve the system performance. In addition, the write data are also allocated to the SSD tier to improve the write performance. Hybrid store [80] uses the regional accessing frequency information through analyzing the historical accessing information to identify the hot data and move these data to SSD tier in order to improve the overall performance.

To fully utilize the advantages of both tiering and caching methods, a new approach that combines the advantages of both methods is proposed to further improve the overall system performance. [101] improves the overall system performance, especially the performance for write intensive workloads with the combination of caching and tiering methods. [102] utilizes the hotness of the data and the load status of the SSD and HDD devices to perform the data allocation and migration. The parallelism performance of all the devices are fully utilized to improve the system response time.

[101] improves the overall system performance, especially the performance for write intensive workloads with the combination of caching and tiering methods. [102] utilizes the hotness of the data and the load status of the SSD and HDD devices to perform the data allocation and migration. The parallel performance of all the devices are fully utilized to improve the system response time. In [103], a new control layer called hybrid storage aware flash translation layer (FDTL) is introduced to

efficiently improves the IO performance and reduces the energy consumption of the storage system. FDTL mechanisms re-distribute the data chunks based on a utility-maximizing policy or minimal-energy-consumption policy. This new distribution takes the advantages of the sequential accessing speed of HDD and random accessing speed of SSD.

To overcome the SSD lifespan limitation problem, some research works use HDD to store the write update data to reduce the number of write operations to SSD. In [107], a new storage architecture called I-CASH is introduced to intelligently couple an array of SSDs and HDDs. In the architecture, the SSD stores the read data, called reference blocks, and HDD stores the difference between the new coming data and the reference blocks, which is called delta. The read request can be handled by combining the reference blocks with the deltas. Alternatively, the write request only needs to update the log of deltas, which minimize the number of write updates to SSD. The performance of the I-CASH architecture is improved by taking the following advantages: fast read speed of SSD, high computing power of processor, and sequential write performance of HDD. With this architecture, the write performance is improved, and the lifespan of SSD is prolonged. However, the data might be lost if the delta is not saved to HDD properly. Thus the delta data in the DRAM should be flushed to disk frequently, and the flush interval is tuned based on the number of dirty delta blocks in the system. A hybrid storage system design called Griffin [108] uses disk drive to cache the write data to SSD. The write requests to SSD are logged into the HDD in sequence and migrated back to SSD when the system is not busy. With such a design, the writes to SSD is minimized and thus the lifespan of SSD is prolonged without significantly impacting on the read performance. However, in these types of architectures, they do not consider the data hotness and the device status when the migration or allocation of the write data to different devices.

Moreover, Researchers also propose a lot of hybrid array structures to efficiently

improve the system performance. [168] explores the parallel processing capability and the hybrid combination of different types of disk drives, such as HDD and SSD, in order to achieve better performance and longer storage system lifespan. A hybrid disk array architecture is proposed in [169] for data-intensive applications, it periodically redistributes the data between the HDD and SSD to balance the performance, energy and reliability. The redistribution considers the changing data access patterns and improves the performance and energy consumption without impairing the reliability. [170] uses the HDD to store the parity data to improve the reliability of the hybrid disk array architecture. [171] proposes a storage array architecture called Herp Rap to improve the failure recovery ability and reliability through copy-on-write snapshot, which is able to recover data to any point-in-time quickly through tracing back the history of block logs.

In this chapter, in addition to allocate the hot write data to HDD in order to prolong the SSD lifespan, the data allocation algorithm proposed also considers the data properties, such as sequential or random, hot read and/or hot write, and the device statuses, such as capacity usage and load status, etc.

6.3 System Architecture Design

To build up a hybrid array storage system with HDDs and SSDs, there are mainly four steps. Firstly, to match the performance of one or few SSDs, a calculated number of HDDs needs to be utilized to form a striped HDD sub-array. Secondly, the striped HDD sub-array is combined together with SSDs to form a hybrid sub-array, which contains both HDDs and SSDs. Thirdly, data allocation and migration algorithms are designed to optimize the data flows between the HDDs and SSDs. The allocation method tries to put the RHD to SSD and WHD to HDD. Lastly, the formulated structure and designed algorithms can be revised when the format

Figure 6.1: IO directions between SSD and HDD

of the hybrid storage system is changed, such as RAID 5/6 or EC [172].

6.3.1 Performance Stripe Formulation

In order to ensure that the HDD array and SSDs have similar sequential performance, the data in HDDs and SSDs will be reformatted in stripes. For example, if the average throughput of an SSD and an HDD are **350Mbps** and **180Mbps** respectively, then at least **2** HDDs are required to form a RAID0 structure to match the sequential performance of one SSD. Alternatively, if the HDD throughput is only around 150Mbps, then 5 HDDs may be required to match the performance of 2 SSDs.

In an HDD device, the performance of Outer Diameter track (OD) is much better compared with that of Inner Diameter track (ID). Meanwhile, the performance of SSD device is much worse when the disk fullness ratio is large compared with the SSD with clean space, as the SSD might need to clean the space to allow new write requests being processed. Due to the performance difference in HDD and SSD devices, a proper performance index shall be assigned to the data stripes of SSD and HDD. Because of the dynamic performance of SSD device under different

HDD1	HDD2	HDD3	SSD1	SSD2	SSD3
P1			P1	P2	P3
P2		P3	P1	P2	P3
P3	P2		P1	P2	P3
P2			P1	P2	P3
P3			P1	P2	P3

Figure 6.2: An example of performance stripes

statuses, a real-time performance monitoring system may be implemented in order to assign the index dynamically. To demonstrate the process of stripe formulation based on different kinds of device statuses, a storage array with **3** HDDs and **3** SSDs is formulated as shown in Fig. 6.2. Let us assume that the **3** SSDs have different throughput μ_1 , μ_2 and μ_3 respectively, with $\mu_1 > \mu_2 > \mu_3$. The partitions inside the SSDs are notated as P_1 , P_2 and P_3 respectively. From the figure, we can notice that the allocation of stripes in HDDs is based on the performance of different zones. The parallel access of stripes from OD zones can be grouped to have similar performance as P_1 in SSDs. Meanwhile, the stripes from ID zones of HDDs only can match the performance of P_3 in SSDs. A specific range of tolerance is allowed for different partitions of the HDDs based on the unequal performance.

Besides the sequential performance consideration during the process of striping, the performance of random access may also need to be considered during the performance index calculation, as the random access to the system can not be fully avoided if the detection and prediction of the incoming IO pattern are not perfect. Therefore, the performance index calculation is defined as in the following equation:

$$f_i(t) = \alpha_1 P_{s,i} / P_{s,Max} + \alpha_2 P_{r,i} / P_{r,Max}. \quad (6.1)$$

In the equation, $P_{s,i}$ and $P_{r,i}$ are sequential performance, which is in terms of throughput and random performance, which is in terms of IOPS, for the particular

stripe i , respectively. $P_{s,Max}$ and $P_{r,Max}$ are the maximum throughput and IOPS performance of stripes from all types of devices in the array. α_1 and α_2 are the weights assigned to the throughput and IOPS in light of the workload properties. Usually, the selection of α_1 and α_2 depends on the proportion of sequential accesses and random accesses in the workload. If the performance difference of the read access and write access is too large, the performance index needs to be calculated independently for them. At that point the overall performance is studied with the consideration of the read and write access patterns of the incoming workload.

6.3.2 Disk Array Construction

All the HDDs and SSDs utilized in the storage system are formed as a disk array. The disk array is further divided into sub-arrays. Each sub-array is the combination of the stripes in SSDs and HDDs. Let us take the example case as shown in Section 6.3.1 to illustrate the process of the disk array construction. In the case, the performance of **1** SSD is similar to the parallel performance of **2** HDDs, which means **2** HDD stripes are required to match the performance of **1** SSD stripe during disk array construction. These constructed sub-arrays can be utilized separately, and they can also be grouped together to form a structure with RAID or EC scheme supported. In Fig. 6.3, a storage architecture with **4** hybrid sub-arrays is presented, whose sub-array contains **2** SSD stripes, and **2** HDD matched stripes. In each sub-array, RAID0 can be applied to improve the data reliability. Meanwhile, RAID5/6 or EC can also be applied to the system when each of the hybrid sub-array is considered as a storage unit.

Figure 6.3: An example of hybrid disk array structure

6.3.3 Data Allocation and Migration

During the processes of data allocation and migration, the decision of the devices to allow the incoming data to be kept has key impact on the overall system performance. A hot data identification algorithm is required to differentiate the hot and cold data, which helps the overall decision. There are two types of cache policies frequently applied in storage systems, which are read-only policy, only caching the hot read data, and write-back policy, caching the hot read and write data. In the cache, the read data is promoted from HDD devices and the write data is normally from external write requests. The hot data identification algorithm normally checks the hotness of the read data based on **ROR** frequency, and the hotness of write data based on WOW frequency. If the same data are accessed frequently by both read and write requests, the data is considered as both **RHD** and **WHD**. Thus, based on the data access properties, they can be categorized as follows:

- $RHD \cap \overline{WHD}$: preferred for SSD
- $\overline{RHD} \cap WHD$: preferred for HDD

- $\overline{RHD} \cap \overline{WHD}$: cold data for both read/write; good for HDD (as HDDs size is much larger than SSD)
- $RHD \cap \overline{WHD}$: an objective function $F(t)$ is designed to determine the choice. $F(t)$ is a time-varying function related to performance, QoS, load balance, remaining disk capacity and etc.

In these categories, $\overline{RHD}$ and $\overline{WHD}$ mean non-RHD and non-WHD. The objective function $F(t)$ for a sub-array j can be defined as $F_j(t)$, which is calculated as following equation:

$$\begin{aligned}
 f_{HDD,j}(t) &= \beta_{11}(R_j(t) - R_{1f,j}) + \beta_{12}(C_{1f,j} - C_j(t)) + \beta_{13}(I_j(t) - I_{1f,j}) \\
 f_{SSD,j}(t) &= \beta_{21}(R_j(t) - R_{2f,j}) + \beta_{22}(C_{2f,j} - C_j(t)) + \beta_{23}(I_j(t) - I_{2f,j}) \\
 F_j(t) &= f_{SSD,j}(t) - f_{HDD,j}(t).
 \end{aligned} \tag{6.2}$$

In the equation, β_{lm} , with $l = 1, 2, 3$ and $m = 1, 2, 3$, are the weighting coefficients. $R_j(t)$ is the response time of external user requests, R_{lf} is the reference response time related to QoS. $C_j(t)$ is the remaining effective capacity of disk, $C_{lf,j}$ is the minimum disk space requirement, $I_{lf,j}$ is the load reference value, $I_j(t)$ is the current load that combines the drive throughput and IOPS, which is calculated based on (6.3).

$$I_j(t) = \gamma_1 \frac{P_{S,j}(t)}{P_{S,max,j}} + \gamma_2 \frac{P_{R,j}(t)}{P_{R,max,j}} \tag{6.3}$$

In (6.3), $P_{S,j}$ and $P_{R,j}$ are the average throughput and IOPS of the HDDs or SSDs in the hybrid sub-array j , respectively. Meanwhile, $P_{s,Max,j}$ and $P_{r,Max,j}$ are the maximum throughput and IOPS of HDDs or SSDs in the sub array j . The

Figure 6.4: A general framework of the data allocation

coefficients γ_m is adjusted for different types of sub-array.

Besides the drive performance, remaining drive space and the load reference value, we can also add the SSD lifespan into consideration when we design $F_{i,j}(t)$ for the stripe i in the sub array j .

The data allocation and migration can be conducted based on the aforementioned data categories. The flowchart for the external write requests is shown in Fig. 6.4. To guarantee the performance of HDDs for the random write requests, whose data can be buffered in a NVM, where the data can be protected even when power is down, and written to disk sequentially in a later time. The flowchart is suitable for both sequential and random requests, and the only difference between them is the definitions of different processes, e.g. RHD , WHD and $F(t)$, etc. For instance, if the random performance is a big concern, then only $\overline{RHD} \cap \overline{WHD}$ data will be directed to HDD as shown in Figure 6.5.

6.3.4 EC/RAID Scheme Consideration

As the performance of HDD and SSD stripes may not perfectly match each other, there will be a synchronization issue. Let us take a Reed-Solomon code $\langle 10, 4 \rangle$ [173] with 10 data and 4 parity structure as an example. If 10 SSD stripes and

Figure 6.5: Data allocation for separated random and sequential accesses

8 HDD stripes, which can be formed as 4 data stripes, are selected, where SSD stripes have exactly the same performance, while the performances of HDD stripes are slightly different, we can have a strict write completion policy, i.e., only after all 14 pieces of data are fully recorded to the corresponding stripes successfully, and then the whole recording process is considered as complete. With this policy, the overall performance might be degraded as it totally depends on the slowest performance stripe. In order to increase the recording speed and make sure the data synchronization, we can slightly change the policy as that if the 10 data and 1 parity are written to the SSD stripes, and the most matched HDD stripes, the system can return the complete signal to a user. The remaining 3 parities can be written to the other 3 HDD data stripes through a non-synchronized way in a later time.

6.4 Model Analysis

To prove the effectiveness of our proposed architecture, we use a simplified analytical model with ideal workload to compare the performance of our proposed architecture with the conventional hybrid storage system and the pure-SSD storage system based on the IOPS, throughput efficiency and SSD lifespan saving.

6.4.1 System Settings

Let us assume the HDD and SSD have the IOPS and throughput properties as shown in Tables 6.1 and 6.2, where the random request size is $s_{rnd} = 4KB$ and sequential request size is $s_{seq} = 512KB$. To match the sequential read or write performance of HDD with SSD, we use **3** HDDs to match with **1** SSD for sequential read performance, and **2** HDDs to match with **1** SSD for sequential write performance. Thus we design a RAID0 sub-array with two cases: **1** SSD and **3** HDDs, **1** SSD and **2** HDDs. In Table 6.2, we assume the HDD uses batch write process for the random write requests, and the IOPS for write requests is based on the performance of HDD with batch write process, which is calculated based on the average seeking time of **5ms**, average rotational latency of **3ms**, and the user request batch size **32**. These write requests can be kept in an NVM, such as PCM, to make sure the data is safe when the disk power is off. Now let us assume the incoming workload has the properties as shown in Table 6.3.

Table 6.1: An example of SSD specification

SSD spec	Read	Write
IOPS (random 4KB)	20K λ_1	10k λ_2
IOPS (sequential 512KB)	20K λ_3	800 λ_4
Throughput (random 4KB)	80MB μ_1	40MB μ_2
Throughput (sequential 512KB)	600MB μ_3	400MB μ_4

Table 6.2: An example of HDD specification

HDD spec	Read	Write
IOPS (random 4KB)	200 λ_5	5K λ_6
IOPS (sequential 512KB)	400 λ_7	400 λ_8
Throughput (random 4KB)	800KB μ_5	20MB μ_6
Throughput (sequential 512KB)	200MB μ_7	200MB μ_8

Table 6.3: Notations of incoming workload properties

Items	values
Sequential ratio	r_s
Read ratio	r_r
Hot R or W ratio	h_w or h_r
Hot R and W ratio	h_{r+w}
Cold R or W ratio	c_r or c_w

6.4.2 Analytical Modeling

From Fig. 6.4 and Fig. 6.5, we know that the requests directed to HDDs include WHD , $\overline{RHD} \cap \overline{WHD}$, and $RHD \cap WHD$, which are allocated to HDDs based on the results of $F_j(t)$ through (6.2). The requests directed to SSD include the RHD , and $RHD \cap WHD$, which are based on the results of (6.2). Now let us assume the percentage of sequential hot read and write data to HDD is x_1 by calculating $F_j(t)$ through (6.2), and percentage of random hot read and write data to HDD is x_2 by calculating $F_j(t)$ through (6.2), we can get the percentages of read/write data to different devices, which are shown in Table 6.4.

Table 6.4: The formulas to calculate the percentages of read and write requests to different devices

Percentage	To HDD	To SSD	Total
Random read r_{rr}	$r_{rr}^{HDD} = (c_r + h_{r+w} \times x_2) \times r_r \times (1 - r_s)$	$r_{rr}^{SSD} = (h_r + h_{r+w} \times (1 - x_2)) \times r_r \times (1 - r_s)$	$r_{rr} = r_{rr}^{HDD} + r_{rr}^{SSD}$
Random write r_{rw}	$r_{rw}^{HDD} = (c_w + h_{r+w} \times x_2 + h_w) \times (1 - r_r) \times (1 - r_s)$	$r_{rw}^{SSD} = (h_{r+w} \times (1 - x_2)) \times (1 - r_r) \times (1 - r_s)$	$r_{rw} = r_{rw}^{HDD} + r_{rw}^{SSD}$
Sequential read r_{sr}	$r_{sr}^{HDD} = (c_r + h_{r+w} \times x_1) \times r_r \times r_s$	$r_{sr}^{SSD} = (h_{r+w} \times (1 - x_1)) \times (1 - r_r) \times r_s$	$r_{sr} = r_{sr}^{HDD} + r_{sr}^{SSD}$
Sequential write r_{sw}	$r_{sw}^{HDD} = (c_w + h_{r+w} \times x_1 + h_w) \times (1 - r_r) \times r_s$	$r_{sw}^{SSD} = (h_{r+w} \times (1 - x_1)) \times (1 - r_r) \times r_s$	$r_{sw} = r_{sw}^{HDD} + r_{sw}^{SSD}$
To device in total	$r_{HDD} = r_{rr}^{HDD} + r_{rw}^{HDD} + r_{sr}^{HDD} + r_{sw}^{HDD}$	$r_{SSD} = r_{rr}^{SSD} + r_{rw}^{SSD} + r_{sr}^{SSD} + r_{sw}^{SSD}$	100%

First, let us assume all the requests are random, and the workload has the properties of $r_r = 40\%$, $h_r = h_w = 90\%$, $h_{r+w} = 5\%$, and $c_r = c_w = 5\%$. And let us assume the random hot read and write data to HDD is $x_2 = r_h = 20\%$. Then we can get the percentages of read and write requests to HDD or SSD as shown in Table 6.5.

Table 6.5: The percentages of random read/write requests to different devices

Percentage	To HDD	To SSD
Read r_{rr}	2.4%	37.6%
Write r_{rw}	57.6%	2.4%

Then, let us assume all the requests are sequential, and the workload has the properties of $r_r = 40\%$, $h_r = h_w = 90\%$, $h_{r+w} = 5\%$, and $c_r = c_w = 5\%$. Let us also assume the hot read and write data to HDD is $x_1 = s_h = 20\%$. Then the percentages of read and write requests to HDD or SSD can be calculated as shown in Table 6.6.

Table 6.6: The percentages of sequential read/write requests to different devices

Percentage	To HDD	To SSD
Read r_{sr}	3.6%	36.4%
Write r_{sw}	59.4%	0.6%

After that, let us assume the percentage of sequential requests is **90%**, and the workload has the properties of $r_r = 40\%$, $h_r = h_w = 90\%$, $h_{r+w} = 5\%$, and $c_r = c_w = 5\%$. Meanwhile, let us assume the sequential hot read and write data to HDD is $x_1 = 80\%$, and random hot read and write data to HDD is $x_2 = 20\%$. Then the percentages of read and write requests to HDD or SSD are calculated and shown in Table 6.7.

The performance efficiency factor of the proposed system can be calculated through (6.1), which needs the IOPS and throughput to be calculated separately. The IOPS and throughput of the proposed system can be calculated through (6.8) and (6.9),

Table 6.7: The percentage of read and write requests to different devices

Percentage	To HDD	To SSD
Sequential read r_{sr}	3.24%	32.76%
Sequential write r_{sw}	53.46%	0.54%
Random read r_{rr}	0.24%	3.76%
Random write r_{rw}	5.76%	0.24%

which are based on the assumption that the ratio of requests does not go beyond the limit of the devices.

$$t_{SSD}^{IOPS} = \frac{1}{\lambda_1} r_{rr}^{SSD} + \frac{1}{\lambda_2} r_{rw}^{SSD} + \frac{1}{\lambda_3} r_{sr}^{SSD} + \frac{1}{\lambda_4} r_{sw}^{SSD} \quad (6.4)$$

$$t_{HDD}^{IOPS} = \frac{1}{\lambda_5} r_{rr}^{HDD} + \frac{1}{\lambda_6} r_{rw}^{HDD} + \frac{1}{\lambda_7} r_{sr}^{HDD} + \frac{1}{\lambda_8} r_{sw}^{HDD} \quad (6.5)$$

$$t_{SSD}^{TP} = \frac{S_{rnd}}{\mu_1} r_{rr}^{SSD} + \frac{S_{rnd}}{\mu_2} r_{rw}^{SSD} + \frac{S_{seq}}{\mu_3} r_{sr}^{SSD} + \frac{S_{seq}}{\mu_4} r_{sw}^{SSD} \quad (6.6)$$

$$t_{HDD}^{TP} = \frac{S_{rnd}}{\mu_5} r_{rr}^{HDD} + \frac{S_{rnd}}{\mu_6} r_{rw}^{HDD} + \frac{S_{seq}}{\mu_7} r_{sr}^{HDD} + \frac{S_{seq}}{\mu_8} r_{sw}^{HDD} \quad (6.7)$$

$$\lambda_{total} = \frac{1}{\max\left(t_{SSD}^{IOPS}, \frac{t_{HDD}^{IOPS}}{N}\right)} \quad (6.8)$$

$$\mu_{total} = \frac{s_{rnd}(r_{rr} + r_{rw}) + s_{seq}(r_{sr} + r_{sw})}{\max\left(t_{SSD}^{TP}, \frac{t_{HDD}^{TP}}{N}\right)} \quad (6.9)$$

$$\lambda_{SSD}^{Pure} = \frac{1}{\frac{1}{\lambda_1} r_{rr} + \frac{1}{\lambda_2} r_{rw} + \frac{1}{\lambda_3} r_{sr} + \frac{1}{\lambda_4} r_{sw}} \quad (6.10)$$

$$\mu_{SSD}^{Pure} = \frac{s_{rnd}(r_{rr} + r_{rw}) + s_{seq}(r_{sr} + r_{sw})}{\frac{s_{rnd}r_{rr}}{\mu_1} + \frac{s_{rnd}r_{rw}}{\mu_2} + \frac{s_{seq}r_{sr}}{\mu_3} + \frac{s_{seq}r_{sw}}{\mu_4}} \quad (6.11)$$

We can calculate the IOPS efficiency factor through:

$$f_{iops} = \lambda_{total} / (2 \times \lambda_{SSD}^{Pure}). \quad (6.12)$$

Meanwhile, we can also compute the throughput efficiency factor through:

$$f_{tp} = \mu_{total} / (2 \times \mu_{SSD}^{Pure}). \quad (6.13)$$

In the above equations, λ is the overall IOPS and μ is the overall throughput of the proposed system. Based on (6.1), and the assumptions that α_1 is the percentage of random requests and α_2 is the percentage of sequential requests, we can get the overall system efficiency, which is shown in Table 6.8.

The SSD lifespan saving can also be calculated through comparing the total size of write requests to SSD in our proposed system and the total size of write requests to SSD in the conventional hybrid and pure SSDs storage system. The SSD lifespan saving also contains two components, the lifespan saving caused by the random write requests ($\lambda_{life} = \frac{\lambda_{HDD}^W}{\lambda_{SSD}^W}$) and the lifespan saving caused by the sequential write requests ($\mu_{life} = \frac{\mu_{HDD}^W}{\mu_{SSD}^W}$). The results of the overall lifespan saving are shown in Table 6.8, where the calculation is based on our proposed example.

$$\begin{aligned} S_{total} = & S_{rnd} \times r_{rw}^{HDD} \times r_r + S_{rnd} \times r_{rw}^{SSD} \times r_r \\ & + S_{seq} \times r_{sw}^{HDD} \times (1 - r_r) + S_{seq} \times r_{sw}^{SSD} \times (1 - r_r), \end{aligned} \quad (6.14)$$

$$S_{total}^{SSD} = S_{rnd} \times r_{rw}^{SSD} \times r_r + S_{seq} \times r_{sw}^{SSD} \times (1 - r_r). \quad (6.15)$$

Table 6.8: The system efficiency and SSD lifespan saving

Item	IOPS efficiency	TP efficiency	SSD lifetime saving (X)
Efficiency (1SSD + 3HDDs vs 2 SSDs)	102.32%	102.32%	99.74
Efficiency (1SSD + 2HDDs vs 2 SSDs)	68.22%	68.22%	99.74

Now we can use $L_{total} = \frac{S_{total}}{S_{total}^{SSD}}$ and the results of (6.15) and (6.14) to calculate the total SSD lifespan saving for the system. Based on the settings in the example, we can get that the SSD lifespan saving is **99.74**×. From which, we can also observe that the SSD lifespan saving does not rely on the number of HDDs matching with **1** SSD, which means the SSD lifespan savings for **2** HDDs matching with **1** SSD and **3** HDDs matching with **1** SSD are the same. When the hybrid sub-array has multiple SSDs, the total write size is shared by more than **1** SSDs. For example, if there are n SSDs in the original all-flash array, and $n - 1$ SSDs in the hybrid sub-array, then there should be m HDDs in the hybrid sub-array to match the original all-flash array when l SSDs are matched with the performance of m HDDs. Then the SSD lifespan saving would be $L_{SSD} = \frac{n}{(n-1)}$.

6.5 Simulation Results

6.5.1 The Impact of Sequential Ratio and Write Ratio of the Workload

In the previous analysis, we have specified the sequential ratio and read request ratio of the incoming workload to illustrate the effectiveness of our proposed system. We fix $h_r = h_w = 90\%$, $h_{r+w} = 5\%$, and $c_r = c_w = 5\%$. Now the system efficiency and SSD lifespan saving under different r_s and r_r are studied and the results are shown in Fig. 6.6.

From the results, we can observe that the smaller the random ratio (larger r_s), the better the system efficiency with respect to IOPS and throughput, and better SSD lifespan saving. This is due to that the performance of sequential read requests with **2–3** HDDs is near to the SSD sequential performance, and sequential write requests with **2–3** HDDs have similar/better performance compared with **1** SSD. And also the percentage of random read requests to HDDs is very small ($< 9\%$) based on the percentage of read requests. We can also observe that the efficiency drops more for the workloads with smaller write ratio when the sequential ratio decreases. This is mainly due to that the random read performance of HDDs is much worse than SSD (**200 vs 20K**). When the sequential ratio decreases, the number of random read requests to HDD increases, which makes the overall performance efficiency worse.

The lifespan saving is mainly contributed by the decreasing of the percentage of write requests to SSD when the sequential ratio of requests increases as the h_{r+w} data has a higher chance to locate to HDD (**80%** for sequential requests).

From the results, we can see that the system efficiency can be up to **120%** with 3 HDDs matching with **1** SSD, and **90%** with **2** HDDs matching with **1** SSD,

6.6.1: efficiency 2 HDDs

6.6.2: efficiency 3 HDDs

6.6.3: lifespan 3 HDDs

Figure 6.6: Efficiency and lifespan savings under different sequential and read ratios.

compared with **2** SSDs. The SSD lifespan saving can be up to **100×** when the incoming requests are mainly sequential requests. The efficiency drops to **30%** and **20%** and lifespan saving drops to **45×** when the incoming requests are mainly read requests and random requests. Thus, we can conclude that our system is more suitable for workload containing more sequential requests or write requests.

6.5.2 The Impact with Threshold Settings for Random and Sequential Properties

Besides the r_s and r_r , we also study the performance and SSD lifespan saving impact of the different percentage of sequential h_{r+w} to HDD and the percentage of random h_{r+w} data to HDD. We fix the parameters $r_s = 90\%$, $r_r = 40\%$, $h_r = h_w = 90\%$, $h_{r+w} = 5\%$, and $c_r = c_w = 5\%$. The results are shown in Fig. 6.7.

From the results, we can observe that the smaller the percentage of sequential and random hot read and write requests to HDDs, the better the system efficiency, however, when the percentage of sequential hot read and write requests to HDD decreases, the SSD lifespan saving becomes worse. This is due to that the performance of SSD is still better than HDDs in respective of sequential read and random read and the number of read requests to SSD increases when the percentage of hot read and write data to HDD is small. However, from the actual value of the efficiency ratio, we can observe that the difference is very small (from **1.08** to **1.01** for **3** HDDs matching with **1** SSD and **0.72** to **0.56** for **2** HDDs matching with **1** SSD) based on different percentage ratios. However, the SSD lifespan can be prolonged from **20×** to nearly **200×**.

6.7.1: efficiency 2 HDDs

6.7.2: efficiency 3 HDDs

6.7.3: lifespan 3 HDDs

Figure 6.7: The performances and lifespan prolonged under different read and write ratios to HDD

6.5.3 The Impact of Hot Read/Write Ratio

We also study the performance impacts of hot data ratio, such as h_r , h_w , and h_{r+w} . We fix $r_r = 40\%$, $c_r = c_w = 5\%$, and $s_h = r_h = 20\%$. The results are shown in Fig. 6.8 and Fig. 6.9. From the figures, we can observe that the hot read ratio and hot read and write ratio have more impact on the overall system performance in terms of IOPS and throughput. This is because only the random read performance has a big differentiation between HDDs and SSD (**200 vs 20K**), and the hot read ratio and hot read and write ratio affect the percentage of random read data to HDD, which affects the overall system performance. SSD lifespan is mainly affected by the hot read and write ratio. This is because only hot read and write data have a chance to be written to SSD, which affects the SSD lifespan most.

In the aforementioned discussion, we only consider the RAID0 performance. When considering the RAID5/6 and EC, we have to take the synchronization into account. For restricted synchronization, the performance is determined by the lowest device. Therefore, the random read requests to the HDD will lower the overall performance. However, the percentage is very low as shown previously. Additionally, we can implement asynchronous access to the disk array to reduce the negative impact of the synchronization process on the system performance.

6.6 System Analysis and Discussion

From the results shown in Section 6.5, we can notice that the proposed hybrid array structure has three benefits compared with the conventional hybrid storage system and storage array comprising pure SSD Flashes. Right off the bat, it can deal with workloads with random or sequential access patterns with comparative performance compared with the storage array containing all Flash, while the overall cost of

6.8.1: efficiency hot read and read/write

6.8.2: lifespan hot read and read/write

6.8.3: efficiency hot read and write

6.8.4: lifespan hot read and write

6.8.5: efficiency hot write and read/write

6.8.6: lifespan hot write and read/write

Figure 6.8: Performance efficiency and lifespan prolonged under different hot read and write ratio under 3:1 HDDs/SSD ratio.

6.9.1: efficiency hot read and read/write

6.9.2: efficiency hot read and write

6.9.3: efficiency hot write and read/write

Figure 6.9: Performance efficiency under different hot read and write ratios under 2:1 HDDs/SSD ratio.

the proposed hybrid array essentially decreases. Furthermore, the SSD lifespan is prolonged enormously, as a large portion of WHD is coordinated to HDD devices. Lastly, the quantity of data migrations between the SSD pool and HDD pool is reduced. Most of the data migration is done inside the same sub-array through interior data bus, which is much faster compared with the network transmission between different storage pools. In general, this new hybrid array structure opens another approach to mix SSDs and HDDs in the hybrid storage system architectures.

In the proposed hybrid array, as we assume the random write requests are handled by HDD devices through batch write process, the performance loss compared with all Flash array is very limited. The main performance reduction is caused by the random read access of data in classification $\overline{RHD} \cap \overline{WHD}$. However, this depends on the assumption that the hot data identification algorithm works impeccably. On the off chance that a portion of the data is arranged wrongly by the identification algorithm, then the overall system performance might decrease as well. Be that as it may, when this circumstance happens, the performance might be far more terrible for the conventional hybrid storage system, which might generate more data migrations between the HDD pool and SSD pool.

From the results shown in Section 6.5, we can also observe that the performance loss of the proposed structure compared with the pure SSD storage array is essentially caused by the random read access data in category $\overline{RHD}$, which is taken care of by HDD device. In any case, the quantity of random read requests to HDD can be controlled by adjusting the RHD definition. For instance, we can bring down the threshold for identifying RHD data to reduce the number of random reads to HDD, which provides a flexibility for clients to tune their storage system in view of the properties of applications, system statuses, and performance requirements. From the analytical results of the hybrid array system that utilizes **2** or **3** HDDs to match with **1** SSD, we can easily conclude that the overall performance efficiency

of the hybrid array system can be effectively maintained, while the SSD lifespan is prolonged up to **100** times with the illustrated workloads.

The performance matching of HDDs with SSD depends on the sequential read or write performance of both sorts of devices. In the event that the incoming workload is from write-dominated application, the device matching is based on the performance of sequential write performance. Under this circumstance, if the sequential read performance is significantly higher than sequential write performance of SSD, the rare sequential read access to HDD may decrease the overall performance. However, we can still properly design the HDD pairs to maintain the overall system QoS.

In the hybrid array system, we have assumed that the random write requests are handled through a batch process, which converts the random write requests to sequential. In this case, the update requests may cause existing data become invalid, and GC process is required to clean these invalid data and save the disk space. However, the GC process consumes disk processing bandwidth, which might reduce the overall system performance. To take care of this issue, we can follow the algorithms utilized in the SMR drive that shown in Chapter 3. When the HDD devices have enough idle periods, where no requests come in during those periods, the devices can handle the GC processes in background. If the number of idle periods is very limited, and the duration is short, foreground GC can be applied to actively clean the dirty data, a specific amount of disk processing bandwidth can be allocated to the foreground GC based on the system performance requirement and disk statuses. In this case, the overall system performance is reduced and we can adjust the performance index of HDDs to guarantee the performances of HDDs and SSD are matched.

In HDD devices, especially the SMR drive, the cost of write update is quite expensive if the batch write process policy is applied. It might be better to divide the whole

devices into a hot area and a cold area. The hot area is only utilized to store the hot write data, and the cold area is utilized to store the cold data, where the update to the cold data is very limited. In this approach, the GC process on the hot data will not influence the cold area, and the number of GC processes can be reduced significantly.

6.7 Conclusion

In this chapter, we propose a new hybrid array architecture to save the system cost and prolong the SSD lifespan, while maintaining the overall system performance. The proposed hybrid array storage structure consists of HDDs and SSDs, which are mixed together in the same performance tier. A new data allocation method is designed to guide the read hot data to SSD and write hot data to HDD whenever possible. The possible extension to RAID5/6 and EC is also discussed. With the proposed structure and algorithm, the overall cost is reduced while the system performance under sequential dominated workloads is maintained. The SSD lifespan can be saved up to **100** times compared with the storage architectures containing pure-SSD or SSD as a write-back cache. Meanwhile, the proposed structure enhances the data reliability as the size of write data to SSD is largely reduced.

In actual physical implementation, the detailed elements, such as the diverse degrees of hotness and randomness should be considered precisely. Under this situation, machine learning techniques can be applied to choose the most appropriate settings. Moreover, the capacity statuses of the devices in the hybrid array are not fully considered, their coefficients are heavily related to the incoming workloads, we will leave it as a future research direction too.

Chapter 7

Hybrid Storage System Performance Improvement through Online Fuzzy Logic Control and Decision Tree

7.1 Introduction

As shown in Chapter 5, hybrid system takes the advantages of high capacity of low tier devices and high performance of high tier devices, to improve the system performance with a limited increment of cost. When designing a hybrid drive, the hot data identification and data migration policies are the most important parts that affect the drive performance. Historically, lots of data identification methods [97] [132] [130], and data migration policies [86] [137] [38] are proposed to improve the hybrid system performance. However, the implementation complexity and high time

consuming of these algorithms make them not fulfilled for the device level design requirements. In this chapter, we focus on improving the data migration policies inside the hybrid drive while keeping the simplicity of the algorithm proposed.

With this in mind, the hot data identification and data migration algorithms need to be designed carefully to make sure that the SSD cache is utilized efficiently. The data migration policy affects the overall performance significantly as it needs to balance the migration benefits (e.g. read hit ratio on the migrated data) and the migration cost (e.g., the overhead of additional cost for the device accesses and data transfer between the devices). However, it is not easy to find one migration policy to suit for different types of workloads with diversified properties. For example, a smaller migration size is suitable for a more random access workload, as it reduces the overhead of additional disk access. However, it is not acceptable for a more locally accessed workload as it reduces the read cache performance. Therefore, it is important to find an algorithm that can adaptively select the most suitable migration policy for various types of workloads.

In order to adaptively adjust the data migration policies for a workload with varied properties, in this chapter, a fuzzy control caching system shown in Fig. 7.1 is proposed, which can automatically select the most suitable migration policy based on the workload properties. The inputs of the fuzzy control caching system are computed based on the moving average of the historical accessing workload properties, such as the request size, r/w ratio, sequential ratio and local ratio, etc. The incoming workload properties are fuzzified as the inputs for the online fuzzy logic process, and the outputs of the fuzzy logic process are defuzzified, and the results are utilized as the inputs for the caching system, which are the caching policies. The outputs of the proposed caching system are the results of system performance, such as the overall response time, read cache hit ratio, read cache hit on SSD ratio and etc. These system performance statistics can be applied as a feedback to the

system, which are utilized together with the inputs to adaptively control the whole caching system.

To verify the effectiveness of the proposed caching policy, various types of hot data identification and data migration algorithms are implemented in the hybrid storage system simulator mentioned in Chapter 5. Meanwhile, different types of workloads with various types of properties are selected for the system usage. All these selected workloads are utilized in the hybrid drive with various types of algorithms implemented, and all the performance statistics are collected. For each type of workload, the cache policy utilized with the best performance is selected, which is the output category for such a workload. Additionally, a decision tree based classification and regression algorithm is implemented with the workload properties as the training inputs and the cache policy categories as the training output to find out the relationships between the workload properties and cache policy categories. These relationships are formulated as fuzzy rules and applied in the proposed caching algorithm.

The main contributions of this chapter are as follows:

- A new online adaptive migration policy with fuzzy logic control is proposed to improve the drive performance under diversified workload properties. The migration policy can adaptively select the most suitable migration size when the workload properties change, which can make the overall drive performance consistent under dynamic workload environments.
- The membership functions and fuzzy rules of the fuzzy controller can be automatically generated through a decision tree classification and regression method based on the workload properties. The decision tree can automatically eliminate the less important workload properties to make sure that the fuzzy rules are simplified and efficient.

- The proposed migration policy is implemented and simulated with a hybrid drive simulator, and the simulation results show the effectiveness of our proposed caching policy under diversified workload environments.

The remainder of this chapter is organized as follows. Section 7.2 lists out the historical research works related to the performance analysis of hybrid drive. Section 7.3 presents the detailed process of the proposed caching system. Section 7.4 compares the performance results between our proposed caching algorithm and the conventional caching policies, and the differences are studied and analyzed. Finally, Section 7.5 draws the conclusion.

7.2 Background

7.2.1 Hot data identification and data migration algorithms

The performance of hybrid drive is affected by the hot data identification and data migration algorithms. There have been lots of research focusing on improving the correctness of hot data identified and migrated. [97] uses a bloom filter to identify the hot data based on the access frequency and recency. A sampling method called HotDataTrap is proposed in [132] to early discard some cold items in the cache to save cache space. [130] applies multiple independent hash functions to reduce the chance of false identification of hot data. All of them are only used to detect the hot data, but optimal data migration size at different time interval is not evaluated. A Hot Random Off-loading method is designed by [38] to allocate the migration files based on the randomness and the hotness, which does not consider the other workload properties. An adaptive look ahead data migration model is proposed by [86] to handle the data migration based on the application specific workload

characteristics, deadline and IO profiles. In [137], a bi-directional migration policy with double thresholds based on feedback is proposed to achieve a balance between storage QoS and migration costs. However, these methods are used for the tiered storage system, which require lots of computational resources, and are not feasible in hybrid drive.

7.2.2 Block level workload characterization

The workload characterization is the first step to design a fuzzy control caching system. In [174], the authors find out that the common characteristics are the disk drive idleness and workload burstiness, and the environment dependent characteristics are response time, request size and arrive rate, etc. The most valuable workload properties that affect the drive performance are extracted out and analyzed by [175]. However, the impact of these workload characteristics on the performance of storage system is not studied in detail. To study the relationship between the workload properties and the drive performance, machine learning approaches can be used, such as the decision tree classification and regression [176], can be applied to extract the main characteristics and tune the impact coefficients. These coefficients can be used to select suitable caching policies based on the training results.

7.2.3 Performance evaluation metrics

To adjust the caching policies based on the drive performance, it is important to measure the performance accurately. Lots of researches only use response time to evaluate the system performance. [177] uses the output delay, which is equivalent to the response time, as the feedback to adjust the system parameters. [178] measures the cache hit/miss ratio and response time under different settings to evaluate their multilevel caching system. [179] calculates the caching miss ratio to evaluate the

network cache performance by their analytical model. [180] proposes to use pre-fetch precision, pre-fetch coverage and pre-fetch pollution to evaluate the system performance under different pre-fetching policies, and select the best suitable pre-fetching policy.

7.2.4 Control algorithms for system performance improvement

The system performance can be optimized through control theory by adjusting the system parameters. Feed-back control is applied in [181] to improve the database cache performance. Feed-back control is also applied by [180] to select the pre-fetching algorithms. [182] compares the proportional control and the fuzzy control on an autonomic computing system and concludes that the fuzzy control does not require the reference inputs to be too specific. Some researchers apply fuzzy control to optimize the performance of system. The appropriate fuzzy rules are selected by [183] to improve the performance of database system. [184] combines the neuro-fuzzy system together with client side caching algorithms to improve the web browser efficiency. But none of them is used inside the hybrid drive to control the caching policies. In this chapter, fuzzy control caching system is designed and applied to adjust the data migration policies in order to improve the hybrid drive performance.

7.3 Methodology

The overall process of the fuzzy control caching system is divided into five steps as shown in Figure 7.2. Firstly, the drive performance for different workloads under different caching policies should be gathered. Secondly, the performance metrics are selected to evaluate the drive performance. Thirdly, the relevant workload properties

Figure 7.1: The fuzzy control block diagram for the caching system

need to be gathered and selected for classification usage. Fourthly, a training process is used to classify the workloads based on the drive performance and workload properties. Lastly, the fuzzy control caching algorithm is designed based on the training results, as shown in Figure 7.1. In the following sections, we will explain the procedures of all the steps in detail.

7.3.1 Gathering of System Performance for Different Kinds of Workloads

To design the fuzzy control caching system, the hybrid drive performances under different caching policies for different types of workloads are gathered first. In order to evaluate the performance, we use the hybrid drive simulator mentioned in Chapter 5, which utilizes HDD as the storage device and SSD as the cache device. Various types of hot data identification and data migration algorithms are implemented in the simulator. With the simulator, we can simulate and gather the hybrid drive performance under different caching policies, especially the data migration policies, for different kinds of workloads.

Figure 7.2: The overall workflow of the fuzzy logic controlled caching system

7.3.2 Evaluation of Different Kinds of Caching Algorithms

In order to accurately evaluate and compare the performance of different caching algorithms with different types of workloads, a performance metric is defined, which includes the overall response time, read/write cache hit ratio, read cache hit on the migration data as shown in Table 7.1. In the performance metric, the response time is the most important factor to evaluate the overall performance of the system, as it covers the benefits of cache hit ratio and drawbacks of disk access and data movement overheads. Based on the overall drive response time, it is able to select the most suitable caching policy for workloads with different properties, and the selection results can be applied in the block level workload classification training process.

7.3.3 Selection of Block Level Workload Characterization

Workload characterization is utilized to calculate the properties of the different types of traces. The characterized results are utilized as the inputs together with the

Table 7.1: The performance metric and description

Name	Description
overall response time	The average response time for all the requests
read cache hit ratio	The number of read requests from SSD cache over the total number of read requests
write cache hit ratio	The number of write requests being hit on the write requests not flushed to disk over the total number of write requests
read hit on prefetch ratio	The number of read requests hit on the migration prefetching part instead of the previous read requests over the total number of read requests

performance metric utilized as the output to the fuzzy logic control cache system. The commonly utilized workload properties include the IO requests arrival rate, inter-arrival time, read/write requests ratio and request size, etc. These properties can be directly extracted from the historical access requests during the online cache process without extra system resources. Additionally, there are complex workload properties that require extra system resource to calculate, such as the sequential ratio, local ratio, local distance, etc. Both categories of workload properties are listed in Table 7.2, where the definitions of all the properties are provided.

In order to gather enough workloads with different properties, we use a synthetic workload generator to generate different kinds of workloads. We can specify the request size, local ratio, sequential ratio and the local distance to generate various types of workloads with diversified properties. Meanwhile, the IOPS and the read/write ratio can also be specified during the generation process. The calculation of average size follows the procedure shown in Algorithm 4, while the local and sequential checking are done through Algorithm 2 and Algorithm 3. Based on the checking results, the number of local and sequential accesses can be counted and the overall local and sequential ratios can be calculated.

Table 7.2: The workload properties and description

Name	Description
Arrive rate	The average number of requests arrived during one second
R/W ratio	The number of read requests or write requests over the number of total requests
Inter arrive time	The average time between two neighbor requests
Avg. read size	The average size of all the read request
Avg. write size	The average size of all the write request
Sequential read ratio	The number of sequential read requests over the total number of read requests
Sequential write ratio	The number of sequential write requests over the total number of write requests
Local read ratio	The number of local read requests where the block number distance is less than a threshold over the total number of read requests
Local read distance	The average distance between the local read requests and their nearest block address previously accessed by a read request

Figure 7.3: Training results with decision tree classification

7.3.4 Process of Training Workload Properties

With the available performance metric and workload characterization methods, all the simulation results generated in section 7.3.1 can be evaluated and the workloads utilized can be characterized. To cross validate the effectiveness of the training results, the generated workloads are split into two or more parts, with first few parts being used as inputs to the simulation with different kinds of caching policies. Then the remaining parts of workloads can be utilized in the validation process.

Based on the performance metric, we can select the most suitable caching policy for each specific workload. With the most suitable caching policies as the training classes, and the workload characteristics as the training inputs, the decision tree classification and regression methods are applied to generate the classes of workload properties for different caching policies. During this process, only the relevant workload properties are utilized as the branches inside the decision tree, while the not

relevant properties are not shown in the tree, which can be eliminated during the fuzzy rule generation process. By reducing these not relevant properties, the online caching algorithm can be simplified as required computation is reduced.

To evaluate the effectiveness of the classification, the remaining part of the workloads can be used as the inputs to verify the training results. The matching of the simulation results and the results of our classification means that the classification is relevant. Otherwise, more workloads might be required during the classification process.

7.3.5 Online Fuzzy Control Caching Algorithm

During the workload training process, the relevant workload properties are selected and classified based on the caching policies, which can be utilized for designing the online fuzzy control caching system. The system design as shown in Figure 7.5, consists of online workload properties collection, fuzzification of workload properties, fuzzy rules generation and usage, defuzzification of fuzzy rule outputs to generate caching policies, and usage of caching policies in the caching system.

7.3.5.1 Workload Properties Calculation

From the training results in Figure 7.3, we can see that the relevant workload properties are the request size, local ratio and sequential ratio. These properties can be calculated by using the logical block number (LBA) and size of the requests. When a new request comes in, the request size and LBA are collected during the workload properties collection process. Based on the request size and LBA, we can check whether the request is local and sequential access through Algorithm 2 and Algorithm 3 and mark the request with *is_local* and *is_sequential* labels for properties calculation usage. Then the moving averages for the request size, local

ratio and sequential ratio are computed through Algorithm 4.

Algorithm 2 Local checking

```

1: procedure CHECK LOCALITY
2:    $n \leftarrow \text{request } id$ 
3:    $\delta \leftarrow \text{local\_threshold}$ 
4:   for  $id \leftarrow \max(0, n - \text{window size})$  to  $n$  do
5:      $x = \text{req}[n].\text{blkno}$ 
6:      $y = \text{req}[id].\text{blkno}$ 
7:     if  $\text{abs}(x - y) < \delta$  then return true
8:     end if
9:   end for
10: end procedure

```

Algorithm 3 Sequential checking

```

1: procedure CHECKING SEQUENTIAL
2:    $n \leftarrow \text{request } id$ 
3:   for  $id \leftarrow \max(0, n - \text{window size})$  to  $n$  do
4:      $x = \text{req}[n].\text{blkno}$ 
5:      $y = \text{req}[id].\text{blkno}$ 
6:     if  $x = y + \text{req}[id].\text{size}$  then return true
7:     end if
8:   end for
9: end procedure

```

7.3.5.2 Fuzzification of Inputs

The fuzzification for each of the relevant workload properties can be easily generated based on the training results shown in Figure 3. For example, the local ratio can be fuzzified into two levels, namely low and high. The request size can be fuzzified into 3 levels, namely low, medium and high. Same as the sequential ratio. The membership functions of all the fuzzifications are shown in Figure 7.4.

7.3.5.3 Fuzzy Rules Generation and Usage

The fuzzy rules can be automatically generated based on the classification branches and the range of workload properties on each branch shown in Figure 7.3. From

Algorithm 4 Average size, local, sequential ratio calculation

```

1: procedure AVERAGE SIZE PROCEDURE
2:    $n \leftarrow \text{request id}$ 
3:    $totalSize \leftarrow 0$ 
4:    $localCount \leftarrow 0$ 
5:    $seqCount \leftarrow 0$ 
6:    $totalCount \leftarrow 0$ 
7:   for  $id \leftarrow \max(0, n - \text{window size})$  to  $n$  do
8:      $totalSize \leftarrow totalSize + req[id].size$ 
9:     if  $req[id].is\_seq$  is true then
10:       $seqCount \leftarrow seqCount + 1$ 
11:    end if
12:    if  $req[id].is\_local$  is true then
13:       $localCount \leftarrow localCount + 1$ 
14:    end if
15:     $totalCount \leftarrow totalCount + 1$ 
16:  end for
17:   $AverageSize \leftarrow totalSize / totalCount$ 
18:   $localRatio \leftarrow localCount / totalCount$ 
19:   $seqRatio \leftarrow seqCount / totalCount$ 
20: end procedure

```

Algorithm 5 Local distance calculation

```

1: procedure LOCAL DISTANCE
2:    $n \leftarrow \text{request id}$ 
3:    $localDistance \leftarrow MAX\_FLT$ 
4:    $\delta \leftarrow local\_threshold$ 
5:   for  $id \leftarrow \max(0, n - \text{window size})$  to  $n$  do
6:      $x = req[n].blkno$ 
7:      $y = req[id].blkno$ 
8:     if  $abs(x - y) < \delta$  then
9:       if  $localDistance > abs(x - y)$  then
10:         $localDistance = abs(x - y)$ 
11:      end if
12:    end if
13:  end for
14:  return  $localDistance$ 
15: end procedure

```

Figure 7.4: Fuzzification of the inputs and outputs of the control system

Figure 7.5: Online fuzzy control caching system overview

Table 7.3: The generated fuzzy rules

sn	Local ratio	Seq ratio	Req size	Outputs
1	small	small	small	X-small
2	small	medium	small	small
3	small	large	small	medium
4	small	small	medium	X-small
5	small	medium	medium	medium
6	small	large	medium	medium
7	small	X	large	X-small
10	large	X	X	large

our example, a **3** dimensional $2 \times 3 \times 3$ fuzzy rule set that consists of **18** fuzzy rules is generated through the automatic generation. The detailed fuzzy rules are shown in Table 7.3, with some common fuzzy rules combined and marked as **X** to simplify the presentation. The utilization of the fuzzy rules in the online cache algorithm just needs to follow the relationships shown in the table. For instance, if the workload properties have a small local ratio, a small sequential ratio and a small average request size, then the fuzzy output will be extremely small, which means the migration size should be small to avoid the overhead of data migration.

7.3.5.4 Defuzzification of Outputs

Based on the fuzzy rules generated and fuzzified input workload properties, we can easily get the fuzzy outputs. The outputs can be transferred to the caching policies through defuzzification process. In our example, the defuzzification process will transfer the fuzzy outputs to the hot data migration sizes, and the data migration sizes are utilized as the input parameters for the online cache algorithm. The defuzzification membership function for our example is shown in Figure 7.4, which is the migration size membership function. From the figure, we can conclude that the migration size is adjusted based on the fuzzy outputs. For instance, if the fuzzy output is small, then the migration size is selected between **0** to **256**.

Table 7.4: Workload properties generation

sn	Workload property name	Settings
1	Request size	8, 32, 64, 96
2	Sequential ratio	0.1, 0.3, 0.5, 0.7, 0.9
3	Local ratio	0.1, 0.3, 0.5, 0.7, 0.9

7.4 Evaluation

To evaluate the drive performance with our proposed caching policy, workloads with diversified properties should be utilized. **100** workloads are generated through a synthetic workload generator and these workloads have different properties, such as local ratio, sequential ratio and request size. The workload properties utilized in the generation process are listed in 7.4. The specifications of HDD and Flash devices utilized in the hybrid drive are same as the ones utilized in Chapter 5, which are listed in Tables 5.1 and 5.2.

7.4.1 Evaluation under Different Workload Properties

To evaluate the performance of our proposed algorithm, the workloads with different sequential ratio and local ratio are combined together to form some mixed workloads. These mixed workloads are applied in the simulator with our proposed algorithm and the conventional caching policies. The simulation results are shown in Figure 7.6. In the figure, the label **10_30** means that a new workload combined with **2** workloads, which have local and sequential ratio of **0.1** and **0.9** respectively. The label ***mixed_5*** means that a workload combined with **5** workloads together, and each of them has local and sequential ratio of **0.1, 0.3, 0.5, 0.7, 0.9** correspondingly. The label **No prefetch** refers to that the cache policy only migrates the exact data required by the incoming workload. The label **Prefetch** means the cache policy allows the migration to move the data before or after the data acquired by the

Figure 7.6: Performances under different workload combinations and caching algorithms

incoming workload with a specified size, which is **256** blocks here. The label **Fuzzy** means that the caching algorithm is our proposed fuzzy control caching algorithm.

From the results, we can see that when the sequential and local ratio difference of the two workloads being mixed together is large, the performance improvement of our proposed algorithm compared with conventional ones is significant. This is on the fact that the fuzzy control caching algorithm can automatically select the best caching policy when the workload properties change. In this case, it can always provide the best performance during the whole period of simulation. However, when the local and sequential difference is not small, the performance improvement is very limited, or even slightly worse than the best performed caching policies. For example, from Figure 7.7, we can notice that the performance of workload **50_70** with our caching algorithm is slightly worse than the caching policy **size=256**. This is due to the fact that the properties of the two workloads utilized to mix are quite similar, where both of them have large local ratio and sequential ratio. Due to the property similarity, the best selected constant data migration size can always provide the best performance. However, our proposed caching policy needs to

Figure 7.7: Performances under different workload combinations and caching algorithms

switch the size of data migration based on the fuzzy rules with workload properties as inputs. The changed policy might not provide the best migration size during the switching period, which might slightly lower the overall performance. In addition to the overall system performance, the cache hit ratio and cache hit on migration data ratio increase a lot compared with caching policy **size=0**, and slightly decrease compared with the caching policy **size=256**. But the disk access overhead decreases a lot compared with the caching policy **size=256**. Meanwhile, the SSD lifespan is saved with our proposed cache policy compared with the caching policy **size=256** as fewer data are migrated to SSD.

7.4.2 Evaluation under Different IOPS

To evaluate the performance under the workloads with different IOPS, we use synthetic workload generator to generate workloads with different sequential and local ratios under various IOPS. These generated workloads with same IOPS are grouped together to form a mixed workload with the specified IOPS. These mixed workloads with different IOPS are utilized in the simulator with our proposed cache policy and

Figure 7.8: Performances under different IOPS and caching algorithms

conventional cache algorithms. The workloads generated have local and sequential ratio of **0.1** and **0.9**, and IOPS from **25** to **250**. The comparison results are shown in Figure 7.8.

From the results, we can notice that the performance of the proposed fuzzy control caching algorithm can be improved by **10–20%**, compared with the best performed caching policy between **size=0** and **size=256** under different IOPS. When the IOPS is larger, the improvement is more significant, since our proposed algorithm can always select the most suitable size of data to migrate to the cache. Thus the benefit of future cache hit and the drawback of the migration cost and disk access cost are balanced. When the local and sequential ratio is small, the proposed policy can automatically reduce the size of data migration, which reduces the disk data transfer time. Meanwhile, if the local and sequential ratio is large, the proposed policy can increase the size of data migration to let more future read requests hit on the cache data. Under both of the scenarios, the disk processing bandwidth is saved, and thus the queuing time is also reduced when the incoming workload has higher IOPS.

7.5 Conclusion

Data migration is one of the key processes to influence the performance of hybrid storage systems. The selection of migration size affects the efficiency of the data migration. Different kinds of workloads might require different sizes of data migration to provide the best performance. Thus it is quite challenging to use one migration policy to fit for various types of workloads. In this chapter, a fuzzy logic control cache algorithm is proposed to dynamically adjust the data migration size based on the changed workload environments. The proposed algorithm improves the overall system performance up to **20%** in most of the cases. Additionally, the fuzzy rules utilized inside the cache algorithms are generated automatically based on the training results of the decision tree classification and regression model. This automation process makes the off-line training process and online cache algorithms be adaptable for different kinds of systems and easily implemented. We shall note that this control algorithm is essentially a closed loop control.

Although the feedback logic controlled caching system improves the system performance under varied workload environment, the system requires a lot of pre-processing, and system performance relies on the decision tree classification results and the generated fuzzy rules. In the future, adaptive control can be directly applied to the caching system to improve the system performance, and make the system adaptively support for different types of workloads. To further simplify the system, adaptive fuzzy control can be utilized to adjust the control parameters for different workload environments and system performance requirements. In addition, we still need to prove the stability and the convergence rate theoretically, which are left as our future work.

Chapter 8

Conclusion and Future Directions

8.1 Conclusion

With the development of IoT and data science technologies, the requirements of data storage systems become more and more challenging. In addition to the normal IOPS and throughput requirements, data reliability, storage capacity, diversified performance requirements are the new aspects that need to be taken into consideration. Hybrid storage architectures are introduced to fulfill these requirements, which provide high accessing speed similar to high tier devices and large storage capacity similar to low tier devices, with minimal cost. In this thesis, the devices utilized in the hybrid storage system, especially the SMR drive, are analyzed through a theoretical model and our developed simulator. Based on that, various types of hybrid storage systems are analyzed and their performances are compared. New designs are proposed on the system architectures and algorithms to enhance the system performance.

In Chapter 3, the performance of the SMR drive under various types of workloads

are studied and analyzed through simulation and theoretical modeling. The maximum IOPS that the SMR drive can support is studied under different kinds of drive parameter settings, such as the write batch size, GC batch size, idle time threshold and disk fullness status, etc. and under various workload properties, e.g. average request size, read and write request ratio, sequential ratio, etc. The SMR drive is modeled as a two-level queue, and the internal queue is modeled as an M/G/1/K queue with state dependent disk processing time. The performance of the queue model is then analyzed through a Markov-chain formulation, and the maximum IOPS the SMR drive can support is analyzed under the constraints of the maximum blocking ratio defined by user and disk fullness status. Meanwhile, a new hybrid GC policy is proposed, which dynamically combines the background GC and foreground GC processes, and provides a consistent drive performance under different disk fullness statuses.

Based on the performance analysis of the SMR drive in Chapter 3, the energy consumption and the disk space cleaning by the GC processes are studied in Chapter 4. To save the drive space utilization, the GC process is required to clean the wasted space caused by the invalid data generated. However, energy consumption is required for the GC process. The trade-off between the disk space cleaned and the energy consumed needs to be considered to find out a suitable GC garbage ratio threshold. A non-linear optimization model is formed to solve the trade-off problem, and the optimal GC garbage ratio threshold is calculated by simplifying the model. The trade-off, which is the benefit of the disk space cleaned minus the cost of energy consumption of GC process is analyzed and simulated through the SMR drive simulator shown in Chapter 3 under various cost ratios, which are the cost of HDD space over the unit price of electricity.

In Chapter 5, various types of hybrid storage systems, such as those based on SSD tiering method, SSD caching method, and SSD hybrid method are studied and

simulated. The performances, including response time, energy consumption, SSD lifespan are analyzed under various kinds of workloads. Meanwhile, the system performances under different kinds of caching algorithms, including read-only, write-back policies are analyzed and compared. These performance comparisons can be utilized as a guide for the hybrid storage system design, and applied as a baseline for analyzing the other kinds of hybrid storage systems.

Based on the performance analysis conducted in Chapter 5, a new type of hybrid storage architecture, called hybrid array is proposed to improve the system performance in Chapter 6. In the hybrid array, the properties of incoming workloads and the statuses of the HDD and SSD devices are considered during the request allocation and data migration. Meanwhile, the different processing power of the SSD and HDD for the read/write requests and random/sequential requests are also taken into consideration during the system design. Sample architecture is provided to show the effectiveness of the new architecture proposed, including the prolonged SSD lifespan and the improved overall IOPS and throughput.

Meanwhile, a new data migration policy is proposed in Chapter 7 to improve the overall system performance. The data migration process in the hybrid storage system is improved through an adaptive fuzzy logic control method to automatically change the migration policy based on the changed workload environments. In addition, the fuzzy rules applied inside the storage systems and the fuzzification for each of trace properties can be automatically generated based on the decision tree classification and regression process.

8.2 Future Directions

In Chapters 3 and 4, the performance and energy consumption of the SMR drive are studied through simulation and theoretical modeling under different kinds of

workload environments. The GC process is analyzed and optimized based on the trade-off between the energy consumption and disk space cleaned. Besides the GC process, there are other types of disk processes to improve the overall drive performance, such as defragmentation, which is to group the data entries with sequential LBA, but non-sequential PBA together to improve the read performance and save the number of address translation entries. In the future, we can take the defragmentation process into consideration when we analyze the SMR drive performance. The cooperation of the GC process and defragmentation process will be also an interesting topic, as the GC process cleans the disk space, while defragmentation may increase the disk space usage, as invalid data may be generated. The trade-off between the GC process and defragmentation needs to be analyzed carefully.

Various types of hybrid storage systems are simulated and compared in Chapter 5. From the results, we can observe that the caching method and tiering method are suitable for different types of workloads. Some researchers also group these two methods together to form a hybrid method to improve the system performance. However, the integration is only done at combination level, which does not consider the detailed advantages and disadvantages of the tiering method and caching method under different kinds of workloads. In fact, the integration can be conducted at a more detailed level, such as a unified structure that treats the cache and tier in a similar way and utilizes the benefits and avoids the drawbacks of the tiering and caching methods. This unified structure can be one of our future directions. At the same time, artificial intelligence and machine learning techniques have shown the capabilities in many industrial applications. Another possible future direction is to investigate the integrated platform with those techniques for a better adaptivity to dynamic workloads and wider applications to various scenarios with tighter performance, reliability, cost, lifespan and energy requirements.

In Chapter 6, a hybrid array is introduced to improve the overall hybrid storage

system with the consideration of workload properties, device status, and parallel processing power. The hybrid array only considers the hybrid storage system with SSDs and HDDs. With the development of emerging memory technologies, new types of hybrid storage systems with other types of memory devices are formed. The proposed hybrid array architecture can also be applied to these new types of memory devices, which have performance and price contrast. For instance, the combination of SSDs and PCMs can be utilized to form a hybrid array, where a specific number of SSDs are utilized to match the performance of one PCM. In the interim, PCM does not have the lifespan issue as SSD, so the algorithms utilized in current hybrid array can be marginally adjusted to fit their properties. Other than the two-level hybrid array, the hybrid array with more levels of devices can likewise be formed, which are all left as our future works.

Currently, our online control theory with machine learning method is tested in a simple storage system by only tuning the parameter of migration size. In a later study, we can improve the algorithm to support for complex storage systems with multiple parameters being selected, which will form a multiple-input multiple-output (*MIMO*) control problem. Besides the migration size, the parameters inside the tiered storage system that can be tuned include: the hot data identification threshold and policy changing, cache replacement policy changing, read data pre-fetching size, and the write cache policy changing, etc. Meanwhile, our current work only focuses on the algorithm design and simulation, but from a theoretical point of view, the stability and convergence rate need to be further studied. Additionally, the storage system can also be modeled as a control theoretic problem. The outputs of the system can be monitored and valued together with the analyzed workload properties to adjust the values of tuning parameters of the storage system in order to meet the design objectives, such as to maximize the system utilization, minimize the access latency while meeting the Total Cost of Ownership (*TCO*) requirement and etc.

Our simulator supports different kinds of storage devices and systems. Based on the simulation, the correctness of storage system modeling can be verified and the effectiveness of designed caching algorithms can be evaluated. The tiered storage system performance improvement through our control theory based caching algorithm is evaluated based on our simulator. However, it is still not easy to find the optimal configuration for a storage system. Moreover, the simulator currently does not support complex storage systems, such as the scale-out software defined storage architecture. One possible future direction is to form an analytical model for the storage systems, and the formulation can be done based on control theory, queuing theory and resource allocation method. The formulated model then can be solved through optimization methods, and the results can be applied to find the most suitable storage system design and parameters selection.

Applied publications in thesis

Journal Papers

1. **Junpeng Niu**, Jun Xu, Lihua Xie, "Hybrid Storage Systems: A Survey of Architectures and Algorithms", IEEE ACCESS, 2018, volume 6, pages 13385-13406.
2. **Junpeng Niu**, Mingzhou Xie, Jun Xu, Li Xia, Lihua Xie, "SMR drive performance analysis under different workload environments", Control Engineering Practice 2018, Volume 75, pages 86-97.
3. **Junpeng Niu**, Jun Xu, Lihua Xie, "New Hybrid Disk Array Structures with Consistent Performance and Prolonged Life", submitted to *IEEE Transactions on Multi-Scale Computing Systems* 2018.

Conference Papers

1. **Junpeng Niu**, Jun Xu, Lihua Xie, "Online Fuzzy Logic Control With Decision Tree For Improving Hybrid Cache Performance", 12th IEEE International Conference on Control and Automation (ICCA), pp. 511-516, 2016.
2. **Junpeng Niu**, Jun Xu, Lihua Xie, "Analytical Modeling of SMR Drive under Different Workload Environments," 13th IEEE International Conference on Control and Automation (ICCA), pp. 1113-1118, 2017.
3. **Junpeng Niu**, Jun Xu, Lihua Xie, "A Deep Look at SMR Performance via Simulation Approach", 13th IEEE International Conference on Control and Automation (ICCA), pp. 713-718, 2017.
4. **Junpeng Niu**, Jun Xu, Lihua Xie, "Optimal Selection of Garbage Collection Dirty Threshold to Balance Power Consumption and Space Release", Chinese Automation Congress (CAC), pp. 6217-6222, 2017.

Patents

1. Jun Xu, **Junpeng Niu**, Jie Yu, William Bernard Boyle, "Hybrid Data Storage Array", US Patent Application No 15/675,018, 10/2017. (Filed)

Other publications

Patents

1. Jun Xu, Wei Xi, Guoxiao Guo, Jianyi Wang, **Junpeng Niu**, “Media Based Cache For Data Storage Device” , US Patent Application No 14/857,467, 10/2015. (Filed)
2. Jun Xu, Shaun Astarabadi, Jie Yu, Grant Mackey, **Junpeng Niu** and Robin O’Neill, “Data Storage System With Multimedia Assets”, US Patent Application No 15/282,837, 10/2017. (Filed)
3. **Junpeng Niu**, Jun Xu, Grant Mackey and Robin O’Neill, “Unified Tier and Cache Structure”, US Patent Application No 16/028387, 07/2018. (Filed)

Conference papers

1. **Junpeng Niu**, Jun Xu, Grant Mackey, Lihua Xie, “Networked storage system simulation and performance analysis”, The 20th IEEE International Conference on High Performance Computing and Communications (HPCC-2018), 2018.

Bibliography

- [1] V. Turner, J. F. Gantz, D. Reinsel, and S. Minton, “The digital universe of opportunities: Rich data and the increasing value of the internet of things,” *IDC Analyze the Future*, 2014.
- [2] Wikipedia, “Flash memory — Wikipedia, the free encyclopedia,” ”https://en.wikipedia.org/wiki/Flash_memory”, 2017.
- [3] Wikipedia, “Solid state drive — Wikipedia, the free encyclopedia,” ”https://en.wikipedia.org/wiki/Solid_state_drive”, 2017.
- [4] N. Muppalaneni and K. Gopinath, “A multi-tier RAID storage system with RAID1 and RAID5,” in *Proceedings of 14th International conference on Parallel and Distributed Processing Symposium, 2000 (IPDPS 2000)*, pp. 663–671, IEEE, 2000.
- [5] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, “Phase change memory,” *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, 2010.
- [6] Y. Huai, “Spin-transfer torque MRAM (STT-MRAM): Challenges and prospects,” *AAPPS Bulletin*, vol. 18, no. 6, pp. 33–40, 2008.
- [7] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, “Metal-oxide RRAM,” *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012.
- [8] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, “Scalable high performance main memory system using phase-change memory technology,” *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 24–33, 2009.

- [9] K. A. Bailey, P. Hornyack, L. Ceze, S. D. Gribble, and H. M. Levy, "Exploring storage class memory with key value stores," in *Proceedings of the 1st Workshop on Interactions of NVM/FLASH with Operating Systems and Workloads (INFLOW)*, pp. 4:1–4:8, ACM, 2013.
- [10] C. W. Smullen, J. Coffman, and S. Gurumurthi, "Accelerating enterprise solid-state disks with non-volatile merge caching," in *Proceedings of IEEE International Conference on Green Computing (IGCC)*, pp. 203–214, IEEE, 2010.
- [11] N. Lu, I.-S. Choi, S.-H. Ko, and S.-D. Kim, "A PRAM based block updating management for hybrid solid state disk," *IEICE Electronics Express*, vol. 9, no. 4, pp. 320–325, 2012.
- [12] M. Tarihi, H. Asadi, A. Haghdoost, M. Arjomand, and H. Sarbazi-Azad, "A hybrid non-volatile cache design for solid-state drives using comprehensive I/O characterization," *IEEE Transactions on Computers*, vol. 65, no. 6, pp. 1678–1691, 2016.
- [13] G. Sun, Y. Joo, Y. Chen, Y. Chen, and Y. Xie, "A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement," in *Emerging Memory Technologies*, pp. 51–77, Springer, 2014.
- [14] W. Xiao, H. Dong, L. Ma, Z. Liu, and Q. Zhang, "HS-BAS: A hybrid storage system based on band awareness of shingled write disk," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pp. 64–71, IEEE, 2016.
- [15] C.-l. Wang, D. Wang, Y. Chai, C. Wang, and D. Sun, "Larger, cheaper, but faster: SSD-SWD Hybrid Storage Boosted by a New SMR-oriented Cache Framework," in *Proceedings of IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, IEEE, 2017.
- [16] Z.-W. Lu and G. Zhou, "Design and implementation of hybrid shingled recording raid system," in *Proceedings of IEEE 14th International Conference on Pervasive Intelligence and Computing (PiCom)*, pp. 937–942, IEEE, 2016.

- [17] D. Luo, J. Wan, Y. Zhu, N. Zhao, F. Li, and C. Xie, "Design and implementation of a hybrid shingled write disk system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 1017–1029, 2016.
- [18] V. Tkachenko, "Flash cache," <https://github.com/facebookarchive/flashcache>, 2016.
- [19] EMC Corporation, "VNX FAST Cache-A Detailed Review," <http://www.emc.com/collateral/software/white-papers/h8046-clariion-celerra-unified-fast-cache-wp.pdf>, 2013.
- [20] D. Reinsel and J. Rydning, "Breaking the 15K-rpm HDD Performance Barrier with Solid State Hybrid Drives," *IDC white paper*, 2013.
- [21] Dell Inc., "Dell Storage SC9000 Array Controller," *Specification Sheet*, 2017.
- [22] B. P. Aviad Offeri, David Hartmanii, "IBM XIV Gen3 Storage System Model 2810-114 120,000 mailbox resiliency exchange 2010 storage solution," 2017.
- [23] NetApp Corporation, "NetApp FAS9000 Modular Hybrid Flash System," 2017.
- [24] Huawei Technologies CO., LTD., "Huawei OceanStor storage cases," *Specification Sheet*, 2017.
- [25] Dell Inc., "Dell EMC Unity Hybrid Storage," *Specification Sheet*, 2017.
- [26] A. Amer, J. Holliday, D. D. Long, E. L. Miller, J.-F. Pâris, and T. Schwarz, "Data management and layout for shingled magnetic recording," *IEEE Transactions on Magnetics*, vol. 47, no. 10, pp. 3691–3697, 2011.
- [27] T. Feldman and G. Gibson, "Shingled magnetic recording: Areal density increase requires new data management," *USENIX*, vol. 38, no. 3, pp. 22–30, 2013.
- [28] G. Gibson and G. Ganger, "Principles of operation for shingled disk devices," *Carnegie Mellon University Parallel Data Lab Technical Report CMU-PDL-11-107*, April 2011.

- [29] W. Liu, L. Zeng, and D. Feng, "APAS: an application aware hybrid storage system combining SSDs and SWDs," in *2016 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pp. 1–4, IEEE, 2016.
- [30] INCITS T10 Technical Committee, "Information technology-zoned block commands (ZBC). draft standard T10/BSR INCITS 536," 2014.
- [31] INCITS T13 Technical Committee, "Zoned-device ATA command set (zac) working draft. draft standard T10/BSR INCITS 536," 2016.
- [32] D. Hall, J. H. Marcos, and J. D. Coker, "Data handling algorithms for autonomous shingled magnetic recording hdds," *IEEE Transactions on Magnet-ics*, vol. 48, no. 5, pp. 1777–1781, 2012.
- [33] R. Pitchumani, A. Hospodor, A. Amer, Y. Kang, E. L. Miller, and D. D. Long, "Emulating a shingled write disk," in *IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 339–346, IEEE, 2012.
- [34] S. Tan, W. Xi, Z. Y. Ching, C. Jin, and C. T. Lim, "Simulation for a shingled magnetic recording disk," *IEEE Transactions on Magnet-ics*, vol. 49, no. 6, pp. 2677–2681, 2013.
- [35] L. Ma and L. Xu, "HMSS: A high performance host-managed shingled storage system based on awareness of SMR on block layer," in *Proceedings of IEEE 18th International Conference on High Performance Computing and Commu-nications (HPCC)*, pp. 570–577, IEEE, 2016.
- [36] S. N. Jones, A. Amer, E. L. Miller, D. D. Long, R. Pitchumani, and C. R. Strong, "Classifying data to reduce long-term data movement in shingled write disks," *ACM Transactions on Storage*, vol. 12, no. 1, p. 2, 2016.
- [37] D. Luo, T. Yao, X. Qu, J. Wan, and C. Xie, "DVS: Dynamic variable-width striping RAID for shingled write disks," in *Proceedings of IEEE International Conference on Networking, Architecture and Storage (NAS)*, pp. 1–10, IEEE, 2016.

- [38] L. Lin, Y. Zhu, J. Yue, Z. Cai, and B. Segee, "Hot random off-loading: A hybrid storage system with dynamic data migration," in *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 318–325, IEEE, 2011.
- [39] X. Chen, W. Chen, Z. Lu, P. Long, S. Yang, and Z. Wang, "A duplication-aware SSD-based cache architecture for primary storage in virtualization environment," *IEEE Systems Journal*, 2015.
- [40] Y. Oh, E. Lee, C. Hyun, J. Choi, D. Lee, and S. H. Noh, "Enabling cost-effective Flash based caching with an array of commodity SSDs," in *Proceedings of the 16th Annual Middleware Conference*, pp. 63–74, ACM, 2015.
- [41] X. Meng, C. Wu, J. Li, X. Liang, Y. Bin, M. Guo, and L. Zheng, "HFA: A hint frequency-based approach to enhance the I/O performance of multi-level cache storage systems," in *Proceedings of 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 376–383, IEEE, 2014.
- [42] Y. Liu, X. Ge, X. Huang, and D. H. Du, "MOLAR: A cost-efficient, high-performance hybrid storage cache," in *Proceedings of International Conference on Cluster Computing (CLUSTER)*, pp. 1–5, IEEE, 2013.
- [43] J. Tai, B. Sheng, Y. Yao, and N. Mi, "SLA-aware data migration in a shared hybrid storage cluster," *Cluster Computing*, vol. 18, no. 4, pp. 1581–1593, 2015.
- [44] S. Huang, Q. Wei, D. Feng, J. Chen, and C. Chen, "Improving flash-based disk cache with lazy adaptive replacement," *ACM Transactions on Storage (TOS)*, vol. 12, no. 2, p. 8, 2016.
- [45] T. Kgil and T. Mudge, "FlashCache: a NAND flash memory file cache for low power web servers," in *Proceedings of the international conference on Compilers, architecture and synthesis for embedded systems (CASES)*, pp. 103–112, ACM, 2006.

- [46] T. Kgil, D. Roberts, and T. Mudge, "Improving NAND Flash based disk caches," in *Proceedings of 35th International Symposium on Computer Architecture (ISCA)*, pp. 327–338, IEEE, 2008.
- [47] Y. Oh, J. Choi, D. Lee, and S. H. Noh, "Caching less for better performance: balancing cache size and update cost of flash memory cache in hybrid storage systems," in *Proceedings of USENIX Conference on File and Storage Technologies (FAST)*, pp. 25:1–25:12, 2012.
- [48] J. Yang, N. Plasson, G. Gillis, and N. Talagala, "HEC: improving endurance of high performance flash-based cache devices," in *Proceedings of the 6th International Systems and Storage Conference (SYSTOR)*, p. 10, ACM, 2013.
- [49] J. Ou, J. Shu, Y. Lu, L. Yi, and W. Wang, "EDM: An endurance-aware data migration scheme for load balancing in SSD storage clusters," in *Proceedings of IEEE 28th International conference on Parallel and Distributed Processing Symposium (IPDPS)*, pp. 787–796, IEEE, 2014.
- [50] R. Appuswamy, D. C. van Moolenbroek, and A. S. Tanenbaum, "Cache, cache everywhere, flushing all hits down the sink: On exclusivity in multilevel, hybrid caches," in *Proceedings of 29th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–14, IEEE, 2013.
- [51] Y. Cheng, W. Chen, Z. Wang, X. Yu, and Y. Xiang, "AMC: an adaptive multi-level cache algorithm in hybrid storage systems," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4230–4246, 2015.
- [52] Y. Chai, Z. Du, X. Qin, and D. A. Bader, "WEC: Improving durability of SSD cache drives by caching write-efficient data," *IEEE Transactions on Computers*, vol. 64, no. 11, pp. 3304–3316, 2015.
- [53] N. Dai, Y. Chai, Y. Liang, and C. Wang, "ETD-Cache: an expiration-time driven cache scheme to make SSD-based read cache endurable and cost-efficient," in *Proceedings of the 12th ACM International Conference on Computing Frontiers (CF)*, p. 26, ACM, 2015.

- [54] F. Ye, J. Chen, X. Fang, J. Li, and D. Feng, "A regional popularity-aware cache replacement algorithm to improve the performance and lifetime of SSD-based disk cache," in *Proceedings of IEEE International Conference on Networking, Architecture and Storage (NAS)*, pp. 45–53, IEEE, 2015.
- [55] H.-P. Chang, S.-Y. Liao, D.-W. Chang, and G.-W. Chen, "Profit data caching and hybrid disk-aware Completely Fair Queuing scheduling algorithms for hybrid disks," *Software: Practice and Experience*, vol. 45, no. 9, pp. 1229–1249, 2015.
- [56] M. Saxena and M. M. Swift, "Design and Prototype of a Solid-State Cache," *ACM Transactions on Storage (TOS)*, vol. 10, no. 3, p. 10, 2014.
- [57] Y. Li, L. Guo, A. Supratak, and Y. Guo, "Enabling performance as a service for a cloud storage system," in *Proceedings of IEEE 7th International Conference on Cloud Computing (CLOUD)*, pp. 554–561, IEEE, 2014.
- [58] Z. Zong, R. Fares, B. Romoser, and J. Wood, "FastStor: improving the performance of a large scale hybrid storage system via caching and prefetching," *Cluster Computing*, vol. 17, no. 2, pp. 593–604, 2014.
- [59] S. Lee, Y. Won, and S. Hong, "Mining-based File Caching in a Hybrid Storage System," *Journal of Information Science and Engineering*, vol. 30, no. 6, pp. 1733–1754, 2014.
- [60] W. Felter, A. Hylick, and J. Carter, "Reliability-aware energy management for hybrid storage systems," in *Proceedings of IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–13, IEEE, 2011.
- [61] K. Bu, M. Wang, H. Nie, W. Huang, and B. Li, "The optimization of the hierarchical storage system based on the hybrid SSD technology," in *Proceedings of Second International Conference on Intelligent System Design and Engineering Application (ISDEA)*, pp. 1323–1326, IEEE, 2012.
- [62] M. Canim, G. A. Mihaila, B. Bhattacharjee, K. A. Ross, and C. A. Lang, "SSD bufferpool extensions for database systems," *Very Large Data Bases Endowment (VLDB)*, vol. 3, no. 1-2, pp. 1435–1446, 2010.

- [63] T. Bisson and S. A. Brandt, "Reducing hybrid disk write latency with flash-backed I/O requests," in *Proceedings of 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 402–409, IEEE, 2007.
- [64] M. Saxena, M. M. Swift, and Y. Zhang, "Flashtier: a lightweight, consistent and durable storage cache," in *Proceedings of the 7th ACM European conference on Computer Systems (EuroSys)*, pp. 267–280, ACM, 2012.
- [65] K. Krish, B. Wadhwa, M. S. Iqbal, M. M. Rafique, and A. R. Butt, "On efficient hierarchical storage for big data processing," in *Proceedings of 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 403–408, IEEE, 2016.
- [66] D. Zhao, K. Qiao, and I. Raicu, "Towards cost-effective and high-performance caching middleware for distributed systems," *International Journal of Big Data Intelligence*, 2015.
- [67] J. Do, D. Zhang, J. M. Patel, D. J. DeWitt, J. F. Naughton, and A. Halverson, "Turbocharging DBMS buffer pool using SSDs," in *Proceedings on International Conference on Management of data (SIGMOD)*, pp. 1113–1124, ACM, 2011.
- [68] D. Lee, C. Min, and Y. I. Eom, "Effective SSD caching for high-performance home cloud server," in *Proceedings of IEEE International Conference on Consumer Electronics (ICCE)*, pp. 152–153, IEEE, 2015.
- [69] S. H. Baek and K.-W. Park, "A fully persistent and consistent read/write cache using flash-based general SSDs for desktop workloads," *Information Systems*, vol. 58, pp. 24–42, 2016.
- [70] Y. Liu, J. Huang, C. Xie, and Q. Cao, "RAF: A random access first cache management to improve SSD-based disk cache," in *Proceedings of IEEE Fifth International Conference on Networking, Architecture and Storage (NAS)*, pp. 492–500, IEEE, 2010.

- [71] STEC Inc., “EnhanceIO open source for Linux,” <https://github.com/stec-inc/EnhanceIO>, 2012.
- [72] D. K. Mridha and L. Bert, “Elastic cache with single parity,” Sept. 1 2015. US Patent 9,122,629.
- [73] HP Inc., “HP SmartCache,” <https://www.hpe.com/us/en/product-catalog/detail/pip.5364342.html>, 2016.
- [74] Y. Liang, Y. Chai, N. Bao, H. Chen, and Y. Liu, “Elastic queue: A universal SSD lifetime extension plug-in for cache replacement algorithms,” in *Proceedings of the 9th ACM International Systems and Storage Conference (SYSTOR)*, pp. 1–11, ACM, 2016.
- [75] G. Yadgar, M. Factor, K. Li, and A. Schuster, “Management of multilevel, multiclient cache hierarchies with application hints,” *ACM Transactions on Computer Systems (TOCS)*, vol. 29, no. 2, p. 5, 2011.
- [76] Z. Zhang, Y. Kim, X. Ma, G. Shipman, and Y. Zhou, “Multi-level hybrid cache: Impact and feasibility,” tech. rep., Oak Ridge National Laboratory (ORNL), 2012.
- [77] Y. Klonatos, T. Makatos, M. Marazakis, M. D. Flouris, and A. Bilas, “Azor: using two-level block selection to improve SSD-based I/O caches,” in *Proceedings of 6th IEEE International Conference on Networking, Architecture and Storage (NAS)*, pp. 309–318, IEEE, 2011.
- [78] F. Chen, D. A. Koufaty, and X. Zhang, “Hystor: making the best use of solid state drives in high performance storage systems,” in *Proceedings of the international conference on Supercomputing (ICS)*, pp. 22–32, ACM, 2011.
- [79] Z. Li, *GreenDM: A versatile tiering hybrid drive for the trade-off evaluation of performance, energy, and endurance*. PhD thesis, Stony Brook University, 2014.
- [80] Y. Kim, A. Gupta, B. Urgaonkar, P. Berman, and A. Sivasubramaniam, “Hybridstore: A cost-efficient, high-performance storage system combining SSDs

- and HDDs,” in *Proceedings of IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 227–236, IEEE, 2011.
- [81] A. Raghavan, A. Chandra, and J. B. Weissman, “Tiera: towards flexible multi-tiered cloud storage instances,” in *Proceedings of the 15th International Middleware Conference*, pp. 1–12, ACM, 2014.
- [82] R. Salkhordeh, H. Asadi, and S. Ebrahimi, “Operating system level data tiering using online workload characterization,” *The Journal of Supercomputing*, vol. 71, no. 4, pp. 1534–1562, 2015.
- [83] J. Hui, X. Ge, X. Huang, Y. Liu, and Q. Ran, “E-HASH: an energy-efficient hybrid storage system composed of one SSD and multiple HDDs,” in *Proceedings of International Conference in Swarm Intelligence*, pp. 527–534, Springer, 2012.
- [84] J. Xue, F. Yan, A. Riska, and E. Smirni, “Storage workload isolation via tier warming: How models can help,” in *Proceedings of 11th International Conference on Autonomic Computing (ICAC)*, pp. 1–11, 2014.
- [85] J. Guerra, H. Pucha, J. S. Glider, W. Belluomini, and R. Rangaswami, “Cost effective storage using extent based dynamic tiering,” in *Proceedings of USENIX Conference on File and Storage Technologies (FAST)*, pp. 20–31, 2011.
- [86] G. Zhang, L. Chiu, and L. Liu, “Adaptive data migration in multi-tiered storage based cloud environment,” in *Proceedings of 3rd International Conference on Cloud Computing (CLOUD)*, pp. 148–155, IEEE, 2010.
- [87] A. Elnably, H. Wang, A. Gulati, and P. J. Varman, “Efficient QoS for multi-tiered storage systems,” in *Proceedings of USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2012.
- [88] H. Wang and P. J. Varman, “Balancing fairness and efficiency in tiered storage systems with bottleneck-aware allocation,” in *Proceedings of USENIX Conference on File and Storage Technologies (FAST)*, pp. 229–242, 2014.

- [89] G. Zhang, L. Chiu, C. Dickey, L. Liu, P. Muench, and S. Seshadri, "Automated lookahead data migration in SSD-enabled multi-tiered storage systems," in *Proceedings of IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–6, IEEE, 2010.
- [90] X. Zhao, Z. Li, and L. Zeng, "FDTM: block level data migration policy in tiered storage system," in *Proceedings of IFIP International Conference on Network and Parallel Computing (NPC)*, pp. 76–90, Springer, 2010.
- [91] H. Shi, R. V. Arumugam, C. H. Foh, and K. K. Khaing, "Optimal disk storage allocation for multi-tier storage system," in *Proceedings of Asia-Pacific Magnetic Recording Conference (APMRC)*, pp. 1–7, IEEE, 2012.
- [92] X. Wu and A. N. Reddy, "Data organization in a hybrid storage system," in *Proceedings of International Conference on Computing, Networking and Communications (ICNC)*, pp. 583–587, IEEE, 2012.
- [93] S. Ma, H. Chen, Y. Shen, H. Lu, B. Wei, and P. He, "Providing hybrid block storage for virtual machines using object-based storage," in *Proceedings of 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 150–157, IEEE, 2014.
- [94] I. Iliadis, J. Jelitto, Y. Kim, S. Sarafijanovic, and V. Venkatesan, "Exaplan: Queueing-based data placement and provisioning for large tiered storage systems," in *Proceedings of IEEE 23rd International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MAS-COTS)*, pp. 218–227, IEEE, 2015.
- [95] X. Wu and A. N. Reddy, "Managing storage space in a Flash and disk hybrid storage system," in *Proceedings of IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems*, pp. 1–4, IEEE, 2009.
- [96] X. Wu and N. R. A., L., "Exploiting concurrency to improve latency and throughput in a hybrid storage system," in *Proceedings of IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 14–23, IEEE, 2010.

- [97] D. Park and D. H. Du, "Hot and cold data identification for Flash memory using multiple bloom filters," in *Proceedings of USENIX Conference on File and Storage Technologies (FAST)*, 2011.
- [98] Y. Lv, B. Cui, X. Chen, and J. Li, "Hotness-aware buffer management for flash-based hybrid storage systems," in *Proceedings of the 22nd ACM international conference on information & knowledge management (CIKM)*, p-p. 1631–1636, ACM, 2013.
- [99] S. H. I. Joseph and S. Roy, "Enhancing tiering storage performance," Mar. 31 2015. US Patent 8,996,808.
- [100] D. Montgomery, "Extent migration for tiered storage architecture," Jan. 7 2014. US Patent 8,627,004.
- [101] S. Hayashi and N. Komoda, "Evaluation of SSD tier method and SSD cache method in tiered storage system," in *Proceedings of the Second Asian Conference on Information Systems (ACIS)*, pp. 8–14, 2013.
- [102] X. Wu and A. N. Reddy, "A novel approach to manage a hybrid storage system.," *JCM*, vol. 7, no. 7, pp. 473–483, 2012.
- [103] S. Bai, J. Yin, G. Tan, Y.-P. Wang, and S.-M. Hu, "FDTL: a unified flash memory and hard disk translation layer," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 4, pp. 1719–1727, 2011.
- [104] K. Oe and K. Okamura, "A hybrid storage system composed of on-the-fly automated storage tiering (OTF-AST) and caching," in *Proceedings of Second International Symposium on Computing and Networking (CANDAR)*, p-p. 406–411, IEEE, 2014.
- [105] K. Oe, T. Nanri, and K. Okamura, "On-The-Fly automated storage tiering with caching and both proactive and observational migration," in *Proceedings of Third International Symposium on Computing and Networking (CANDAR)*, pp. 371–377, IEEE, 2015.

- [106] M. Abashkin, A. Natanzon, and E. Bachmat, "Integrated caching and tiering according to use and QoS requirements," in *Proceedings of IEEE 34th International Performance on Computing and Communications Conference (IPCCC)*, pp. 1–8, IEEE, 2015.
- [107] Q. Yang and J. Ren, "I-CASH: Intelligently coupled array of SSD and HDD," in *IEEE 17th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 278–289, IEEE, 2011.
- [108] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, "Extending SSD lifetimes with disk-based write caches," in *Proceedings of USENIX Conference on File and Storage Technologies (FAST)*, pp. 101–114, 2010.
- [109] C. Li, D. Feng, Y. Hua, and F. Wang, "Improving RAID performance using an endurable SSD cache," in *Proceedings of 45th International Conference on Parallel Processing (ICPP)*, pp. 396–405, IEEE, 2016.
- [110] W. Xiao, X. Lei, R. Li, N. Park, and D. J. Lilja, "PASS: a hybrid storage system for performance-synchronization tradeoffs using SSDs," in *Proceedings of IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, pp. 403–410, IEEE, 2012.
- [111] M. K. Aguilera, K. Keeton, A. Merchant, K.-K. Muniswamy-Reddy, and M. Uysal, "Improving recoverability in multi-tier storage systems," in *Proceedings of 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 677–686, IEEE, 2007.
- [112] B. Mao, H. Jiang, S. Wu, Y. Fu, and L. Tian, "SAR: SSD assisted restore optimization for deduplication-based storage systems in the cloud," in *Proceedings of 7th International Conference on Networking, Architecture and Storage (NAS)*, pp. 328–337, IEEE, 2012.
- [113] C. Wu, X. He, Q. Cao, and C. Xie, "Hint-k: An efficient multi-level cache using k-step hints," in *Proceedings of 39th International Conference on Parallel Processing*, pp. 624–633, IEEE, 2010.

- [114] T. M. Wong and J. Wilkes, “My cache or yours? making storage more exclusive,” in *Proceedings of USENIX Annual Technical Conference, General Track*, pp. 161–175, 2002.
- [115] B. S. Gill, “On multi-level exclusive caching: offline optimality and why promotions are better than demotions,” in *Proceedings of USENIX Conference on File and Storage Technologies (FAST)*, USENIX Association, 2008.
- [116] T. Bisson and S. A. Brandt, “Flushing policies for NVCache enabled hard disks,” in *Proceedings of Mass Storage Systems and Technologies (MSST)*, pp. 299–304, IEEE, 2007.
- [117] L. Shi, J. Li, C. Jason Xue, and X. Zhou, “Hybrid nonvolatile disk cache for energy-efficient and high-performance systems,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 1, p. 8, 2013.
- [118] T.-C. Huang and D.-W. Chang, “Tridentfs: a hybrid file system for non-volatile ram, flash memory and magnetic disk,” *Software: Practice and Experience*, vol. 46, no. 3, pp. 291–318, 2016.
- [119] H. Kim, S. Seshadri, C. L. Dickey, and L. Chiu, “Evaluating phase change memory for enterprise storage systems: A study of caching and tiering approaches,” *ACM Transactions on Storage (TOS)*, vol. 10, no. 4, pp. 15:1–15:21, 2014.
- [120] G. Dhiman, R. Ayoub, and T. Rosing, “PDRAM: a hybrid PRAM and DRAM main memory system,” in *Proceedings of ACM/IEEE 46th Design Automation Conference (DAC)*, pp. 664–669, IEEE, 2009.
- [121] Wikipedia, “Cache replacement policies — Wikipedia, the free encyclopedia,” ["https://en.wikipedia.org/wiki/Cache_replacement_policies"](https://en.wikipedia.org/wiki/Cache_replacement_policies), 2017.
- [122] L. Cherkasova, *Improving WWW proxies performance with greedy-dual-size-frequency caching policy*. Hewlett-Packard Laboratories, 1998.
- [123] E. J. O’neil, P. E. O’neil, and G. Weikum, “The lru-k page replacement algorithm for database disk buffering,” *ACM SIGMOD Record*, vol. 22, no. 2, pp. 297–306, 1993.

- [124] S. Jin and A. Bestavros, "Popularity-aware greedy dual-size web proxy caching algorithms," in *Proceedings of 20th international conference on Distributed computing systems (ICDCS)*, pp. 254–261, IEEE, 2000.
- [125] S. Rajasekaran, S. Duan, W. Zhang, and T. Wood, "Multi-cache: Dynamic, Efficient Partitioning for Multi-tier Caches in Consolidated VM Environments," in *Proceedings of IEEE International Conference on Cloud Engineering (IC2E)*, pp. 182–191, IEEE, 2016.
- [126] Y. Letian, H. Chong Ethan, and Z. Liu, "Solid-state disk caching the top-k hard-disk blocks selected as a function of access frequency and a logarithmic system time," Sept. 16 2014. US Patent 8,838,895.
- [127] A. Schaefer and M. Gries, "Adaptive address mapping with dynamic runtime memory mapping selection," 2015. US Patent 9,026,767.
- [128] X. Zhao, Z. Li, and L. Zeng, "A hierarchical storage strategy based on block-level data valuation," in *Proceedings of Fourth International Conference on Networked Computing and Advanced Information Management (NCM)*, vol. 1, pp. 36–41, IEEE, 2008.
- [129] D. Park, B. Debnath, Y. Nam, D. H. Du, Y. Kim, and Y. Kim, "HotData-Trip: a sampling-based hot data identification scheme for flash memory," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC)*, pp. 1610–1617, ACM, 2012.
- [130] J.-W. Hsieh, L.-P. Chang, and T.-W. Kuo, "Efficient on-line identification of hot data for Flash-memory management," in *Proceedings of the ACM symposium on Applied computing*, pp. 838–842, ACM, 2005.
- [131] Y. Lv, X. Chen, G. Sun, and B. Cui, "A probabilistic data replacement strategy for flash-based hybrid storage system," in *Proceedings of Asia-Pacific Web Conference (APWeb)*, pp. 360–371, Springer, 2013.
- [132] D. Park, Y. J. Nam, B. Debnath, D. H. Du, Y. Kim, and Y. Kim, "An on-line hot data identification for Flash-based storage using sampling mechanism," *ACM SIGAPP Applied Computing Review*, vol. 13, no. 1, pp. 51–64, 2013.

- [133] M. Alshawabkeh, A. Riska, A. Sahin, and M. Awwad, "Automated storage tiering using Markov Chain correlation based clustering," in *Proceedings of 11th International Conference on Machine Learning and Applications (ICMLA)*, pp. 392–397, IEEE, 2012.
- [134] X. Meng, L. Zheng, L. Li, and J. Li, "PAM: An efficient power-aware multi-level cache policy to reduce energy consumption of software defined network," in *Proceedings of 1st International Conference on Industrial Networks and Intelligent Systems (INISCom)*, pp. 18–23, IEEE, 2015.
- [135] J. Tai, B. Sheng, Y. Yao, and N. Mi, "Live data migration for reducing SLA violations in multi-tiered storage systems," in *Proceedings of International Conference on Cloud Engineering (IC2E)*, pp. 361–366, IEEE, 2014.
- [136] M. He, L. Xing, and G. Li, "A data migration strategy for hsm based on data value," *JOURNAL OF INFORMATION & COMPUTATIONAL SCIENCE*, vol. 8, no. 2, pp. 312–317, 2011.
- [137] X. Zhao, Z. Li, X. Zhang, and L. Zeng, "Block-level data migration in tiered storage system," in *Proceedings of Second International Conference on Computer and Network Technology (ICCNT)*, pp. 181–185, IEEE, 2010.
- [138] Y. Cheng, F. Douglass, P. Shilane, G. Wallace, P. Desnoyers, and K. Li, "Erasing Beladys limitations: In search of Flash cache offline optimality," in *Proceedings of USENIX Annual Technical Conference (USENIX ATC)*, 2016.
- [139] W. Burkhard and J. Palmer, "Rotational position optimization (RPO) disk scheduling," tech. rep., University of California at San Diego, La Jolla, CA, USA, 2001.
- [140] D. M. Jacobson and J. Wilkes, *Disk scheduling algorithms based on rotational position*. Palo Alto, CA: Hewlett-Packard Laboratories, 1991.
- [141] P. J. Denning, "Effects of scheduling on file memory operations," in *1967 Proceedings of spring joint computer conference*, pp. 9–21, ACM, April 1967.
- [142] Y. Cho and T. Kim, "An efficient scheduling algorithm for NCQ within SSDs," *IEICE Electronics Express*, vol. 12, no. 4, pp. 20150066–20150066, 2015.

- [143] B. Jun and D. Shin, "Workload-aware budget compensation scheduling for NVMe solid state drives," in *Proceedings of Non-Volatile Memory System and Applications Symposium (NVMSA)*, pp. 1–6, IEEE, 2015.
- [144] H.-P. Chang, S.-Y. Liao, D.-W. Chang, and G.-W. Chen, "Profit data caching and hybrid disk-aware completely fair queuing scheduling algorithms for hybrid disks," *Software: Practice and Experience*, 2014.
- [145] C. Gao, L. Shi, M. Zhao, C. J. Xue, K. Wu, and E. H. Sha, "Exploiting parallelism in I/O scheduling for access conflict minimization in flash-based solid state drives," in *Proceedings of 30th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–11, IEEE, 2014.
- [146] Z. Yao, I. Papapanagiotou, and R. D. Callaway, "Multi-dimensional scheduling in cloud storage systems," in *Proceedings of International Communications Conference (ICC)*, 2015.
- [147] A. S. Lebrecht, N. J. Dingle, and W. J. Knottenbelt, "A performance model of zoned disk drives with I/O request reordering," in *2009 Sixth International Conference on the Quantitative Evaluation of Systems (QEST)*, pp. 97–106, IEEE, 2009.
- [148] E. Shriver, A. Merchant, and J. Wilkes, "An analytic behavior model for disk drives with readahead caches and request reordering," *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 1, pp. 182–191, 1998.
- [149] C. Ruemmler and J. Wilkes, "An introduction to disk drive modeling," *Computer*, vol. 27, pp. 17–28, Mar. 1994.
- [150] M. Xie, J. Xu, and L. Xia, "Performance analysis of media-based cache in hard disk drives," in *IEEE Region Ten Conference (TENCON)*, IEEE, 2016.
- [151] M. Seltzer, P. Chen, and J. Ousterhout, "Disk scheduling revisited," in *Proceedings of the Winter 1990. USENIX*, pp. 313–324, Washington, DC, 1990.
- [152] B. L. Worthington, G. R. Ganger, and Y. N. Patt, "Scheduling algorithms for modern disk drives," *ACM SIGMETRICS Performance Evaluation Review*, vol. 22, no. 1, pp. 241–251, 1994.

- [153] D. Gross, *Fundamentals of queueing theory*. John Wiley & Sons, 2008.
- [154] E. Altman, K. Avrachenkov, C. Barakat, and R. Núñez-Queija, “State-dependent M/G/1 type queueing analysis for congestion control in data networks,” *Computer Networks*, vol. 39, no. 6, pp. 789–808, 2002.
- [155] H. Abouee-Mehrizi and O. Baron, “State-dependent M/G/1 queueing systems,” *Queueing Systems*, vol. 82, no. 1-2, pp. 121–148, 2016.
- [156] J. Niu, J. Xu, and L. Xie, “Analytical modeling of SMR drive under different workload environments,” in *Proceedings of 13th IEEE International Conference on Control and Automation (ICCA)*, pp. 1113–1118, IEEE, 2017.
- [157] S. Chen and D. Towsley, “The design and evaluation of RAID 5 and parity striping disk array architectures,” *Journal of Parallel and Distributed Computing*, vol. 17, no. 1-2, pp. 58–74, 1993.
- [158] H. A. David and H. Nagaraja, *Order Statistics*. John Wiley & Sons, 2004.
- [159] E. Cinlar, *Introduction to stochastic processes*. Courier Corporation, 2013.
- [160] L. Kleinrock, *Queueing systems, volume 2: Computer applications*, vol. 66. wiley New York, 1976.
- [161] S. Gougeaud, S. Zertal, J.-C. Lafoucriere, and P. Deniel, “A generic and open simulation tool for large multi-tiered hierarchical storage systems,” in *Proceedings of International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pp. 1–8, IEEE, 2016.
- [162] Y. Yamato, “Use case study of HDD-SSD hybrid storage, distributed storage and HDD storage on openstack,” in *Proceedings of the 19th International Database Engineering & Applications Symposium*, pp. 228–229, ACM, 2015.
- [163] C. San-Lucas and C. L. Abad, “Towards a fast multi-tier storage system simulator,” in *Proceedings of IEEE Ecuador Technical Chapters Meeting (ETCM)*, pp. 1–5, IEEE, 2016.

- [164] A. Varga *et al.*, “The OMNeT++ discrete event simulation system,” in *Proceedings of the European simulation multiconference (ESM)*, vol. 9, pp. 65–71, 2001.
- [165] Storage Performance Council, “SPC BENCHMARK 1C (SPC-1C) SPC BENCHMARK 1C/ENERGY EXTENSION (SPC-1C/E) OFFICIAL SPECIFICATION,” 2012.
- [166] Application, OLTP, “I/O and search engine I/O. umass trace repository,” 2007.
- [167] R. T. Collins, A. J. Lipton, T. Kanade, H. Fujiyoshi, D. Duggins, Y. Tsin, D. Tolliver, N. Enomoto, O. Hasegawa, P. Burt, *et al.*, “A system for video surveillance and monitoring,” *VSAM final report*, pp. 1–68, 2000.
- [168] S. Yin, Z. Ding, X. Ruan, X. Zhu, Z. Tang, K. Li, and X. Qin, “Hcdd: The hybrid combination of disk drives in active storage systems,” in *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESSE), 2015 IEEE 17th International Conference on*, pp. 678–683, IEEE, 2015.
- [169] T. Xie and Y. Sun, “Dynamic data reallocation in hybrid disk arrays,” *IEEE Transactions on Parallel & Distributed Systems*, no. 9, pp. 1330–1341, 2009.
- [170] B. Mao, H. Jiang, S. Wu, L. Tian, D. Feng, J. Chen, and L. Zeng, “Hpda: A hybrid parity-based disk array for enhanced performance and reliability,” *ACM Transactions on Storage (TOS)*, vol. 8, no. 1, p. 4, 2012.
- [171] L. Zeng, D. Feng, B. Mao, J. Chen, Q. Wei, and W. Liu, “Herprap: A hybrid array architecture providing any point-in-time data tracking for datacenter,” in *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, pp. 311–319, IEEE, 2012.
- [172] H. Weatherspoon, J. Kubiatowicz, *et al.*, “Erasure coding vs. replication: A quantitative comparison,” in *Proceedings of International Workshop on Peer-to-Peer Systems (IPTPS)*, vol. 1, pp. 328–338, Springer, 2002.

- [173] S. B. Wicker and V. K. Bhargava, *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.
- [174] A. Riska and E. Riedel, "Disk drive level workload characterization," in *USENIX Annual Technical Conference, General Track*, vol. 2006, pp. 97–102, 2006.
- [175] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda, "Characterization of storage workload traces from production windows servers," in *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*, p-p. 119–128, IEEE, 2008.
- [176] L. Rokach and O. Maimon, "Decision trees," in *Data mining and knowledge discovery handbook*, pp. 165–192, Springer, 2005.
- [177] Y. Lu, T. Abdelzaher, C. Lu, L. Sha, and X. Liu, "Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers," in *Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE*, pp. 208–217, IEEE, 2003.
- [178] Y. Zheng, B. T. Davis, and M. Jordan, "Performance evaluation of exclusive cache hierarchies," in *Performance Analysis of Systems and Software, 2004 IEEE International Symposium on-ISPASS*, pp. 89–96, IEEE, 2004.
- [179] M. Gallo, B. Kauffmann, L. Muscariello, A. Simonian, and C. Tanguy, "Performance evaluation of the random replacement policy for networks of caches," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, pp. 395–396, ACM, 2012.
- [180] Y. Chen, H. Zhu, H. Jin, and X.-H. Sun, "Algorithm-level feedback-controlled adaptive data prefetcher: Accelerating data access for high-performance processors," *Parallel Computing*, vol. 38, no. 10, pp. 533–551, 2012.
- [181] R. K. Gullapalli, C. Muthusamy, A. V. Babu, and R. N. Marndi, "A feedback control solution in improving database driver caching," *International Journal of Engineering Science and Technology*, vol. 3, no. 7, 2011.

- [182] H. S. Venkatarama and K. C. Sekaran, "Regulating response time in an autonomic computing system: A comparison of proportional control and fuzzy control approaches," *arXiv preprint arXiv:1011.1738*, 2010.
- [183] S. Rodd, U. P. Kulkarni, and A. Yardi, "Fuzzy controlled architecture for performance tuning of database management system," *International Journal of Computer Applications*, vol. 39, no. 5, pp. 1–5, 2012.
- [184] W. Ali and S. M. Shamsuddin, "Integration of least recently used algorithm and neuro-fuzzy system into client-side web caching," *International Journal of Computer Science and Security (IJCSS)*, vol. 3, no. 1, p. 1, 2009.