Techniques for Optimizing Cost of Enterprise Data Management

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Nagapramod Mandagere

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

David H.C. Du, Advisor

November, 2015

© Nagapramod Mandagere 2015
ALL RIGHTS RESERVED

Acknowledgements

Numerous people over the years have positively impacted me and my work. I owe a great deal of gratitude towards all of them. First, I would like to thank Professor David Du for being my advisor and mentor not just during this work but my entire professional career. He has had a big hand in shaping me into what I am today. Working remotely can be challenging given time zone and other issues. But for the flexibility and accommodation by thesis committee members - Prof. David Du, Prof. Jon Weissman, Prof. Mohamed Mokbel and Prof. John Sartori, I could not have made it this far. A very special thanks to all of you.

Over the years I have had the opportunity to work with a large number of people from different backgrounds both at school and during my work at IBM Research. I would like to thank Sandeep Uttamchandani for giving me my first break. He has been a constant source for inspiration over the years. I would like to thank my uncle Prof. Suresh Mukhnahallipatna for being my mentor for life. Have always looked up to him whenever I needed inspiration/guidance. Collaboration is a key attribute to success. Over the years I have had the opportunity to collaborate with a lot of helpful researchers. I would like to thank Ramani Routray, Pin Zhou, Yang Song, Gabriel Alatorre, Mu Qiang, Rakesh Jain, Paul Bradshaw, Mirza Baig, Jim Diehl.

Balancing work, grad school and life can be challenging. I could not have done it without awesome support from my managers over the years - Sandeep Gopisetty, Ramani Routray, Heiko Ludwig. Without constant encouragement and accommodation from them it would not have been possible. Thanks to IBM Research in general for naturing research and researchers.

Last but not the least, I would like to thank my family for their unwavering support through the years. Without their persistence and support, I could not have made it

this far. Special thanks to my wife/my better half Swetha for continuing to believe in me, my work and my goals. I would like to thank my parents for not losing hope and continuing to encourage and support me through the years. I would like to thank all my relatives and friends for their support and encouragement.

Dedication

To my family...

Abstract

Rapid adoption of data driven decision making, Internet of Things(IoT), mass digitization of content have led to unprecedented changes in data storage requirements. From the typical paradigms of online, nearline and offline data, now the boundaries are increasingly blurred with more fuzzy interactions between realtime transactions and analytics workloads. Optimizing Total Cost of Ownership (TCO) of data storage infrastructure characterized by capital/acquisition costs and operational management costs necessitate a radical redesign of data storage infrastructure to cope with exponential data growth not just in terms of capacity, but also veracity and velocity of data. Towards this goal, we make the following key contributions, - **a) Improving operational efficiency through application aided storage power management:** Energy consumption of the storage solutions contributes significantly to the operational efficiency of data management. We propose a storage solution called GreenStor, centered on MAID, but with more scalable and efficient data movement to aid in energy conservation based on extent-based cache management. **b) Improving operational efficiency of data protection through model based approaches:** Operational inefficiencies and scalability issues in data protection systems mainly stem from the usage of static policy based management. Using a data driven approach we characterize these inefficiencies and propose a model based dynamic backup scheduling framework that attempts to address key scalability and performance limitations of current backup systems. **c) Improving cost efficiencies of data protection using commodity Software Defined Storage(SDS):** Continuous Data Protection (CDP) enables recoverability to any point in time (time travel) facilitated via journaling of every write made by a system to disk. We propose cCDP - a Cloud CDP framework that efficiently combines cloud object stores with edge caching to address requirements of low cost, high capacity, low latency and high storage throughput. Operational efficiency and usability of CDP is a function of how efficiently data can be restored in case of a failure. To address recovery requirements, we propose a novel method of organizing the layout of CDP logs on object storage to optimize temporal search and an object naming encoding scheme coupled with a Trie based queueing mechanism to optimize spatial search.

Contents

Acknowledgements **i**

Dedication **iii**

Abstract **iv**

List of Tables **ix**

List of Figures **x**

1 Introduction **1**

1.1 Research Challenges 2

1.2 Scope & Contributions 5

1.2.1 Improving operational efficiency thru application aided storage power management 5

1.2.2 Improving operational efficiency and scalability of data protection through model based approaches 6

1.2.3 Improving cost efficiencies of data protection using commodity Software Defined Storage (SDS) 6

1.2.4 Improving operational efficiency through smart restore optimization using commodity SDS 6

1.3 Organization 7

2 Background & Related Work **8**

2.1 Datacenter Power Usage 8

2.2 Storage Power Usage 9
2.3 Taxonomy of Storage Power Optimization Techniques 11
2.3.1 Adaptive spin down & Dynamic modulation of rotation speed . . 11
2.3.2 Optimizing Data Layouts 13
2.3.3 Caching for Power Optimization 14
2.3.4 RAID Adaptations 15
2.3.5 Emerging Media Types 16
2.4 Data Protection 18
2.5 Taxonomy of Data Protection Technologies 19
2.5.1 Appliance based Backup 19
2.5.2 Continuous Data Protection(CDP) 22
2.5.3 Snapshotting & Mirroring 25
2.6 Relative Efficiencies of Data Protection 26
2.7 Software Defined Storage & Cost Efficiencies 27

3 Application-Aided Energy-Efficient Storage 29
3.1 GreenStor Storage System Architecture 33
3.1.1 Virtualization Mechanism 33
3.1.2 Application Hint Specification 35
3.1.3 Data/Cache Layout 36
3.2 Extent-based Metadata Management 37
3.2.1 Monitoring Access Patterns 39
3.2.2 Extent Allocation 40
3.2.3 Global Mapping 41
3.3 Deadline-based Prefetch Scheduler 43
3.3.1 Opportunistic Deep Prefetcher 46
3.3.2 Delayed/Lazy Execution 48
3.4 Preliminary Results 51
3.4.1 Comparison of Scheduling Schemes 51
3.4.2 Comparison of Metadata Management 54
3.5 Conclusion & Future Work 56

4 Backup Characterization **57**
4.1 Data Collection 59
4.2 Configuration Variety 60
4.2.1 Storage Variety 60
4.2.2 Association Variety 62
4.2.3 Backup Window Variety 63
4.3 Workload Variety 65
4.3.1 Backup Server Ingest Variation 65
4.3.2 Backup Type Variation 66
4.3.3 Client Workload Variation 67
4.4 Discussion 68

5 Model based Backup Scheduling **71**
5.1 Performance Variability 72
5.2 Backup Optimization 75
5.2.1 Model based Dynamic Scheduling 75
5.3 Trace driven Simulation 77
5.3.1 Calibration & Sensitivity Analysis 78
5.4 Experimental Evaluation 78
5.4.1 Modeling Accuracy 79
5.4.2 Scheduling Performance 79
5.5 Conclusions & Future Work 80

6 Cloud object based Continuous Data Protection **82**
6.1 Feasibility Analysis 84
6.1.1 Ingestion Behavior 85
6.1.2 Restore/Recovery Behavior 86
6.2 cCDP System Design 87
6.2.1 Overall Architecture 88
6.2.2 Destager 89
6.2.3 Ordering & Error Recovery 90
6.3 Performance Evaluation 91
6.3.1 Experimental Setup 91

6.3.2 Results 92
6.4 Conclusions & Future Work 93

7 Recovery Optimizations for Cloud Continuous Data Protection 95

7.1 Object Storage Retrieval Characteristics 96
7.2 Data Layout Optimization for Full System Recovery 98
7.3 Naming optimizations for Individual File/Subtree Recovery 103
7.3.1 Exploiting Naming Semantics 103
7.3.2 Semantic Coalescing 104
7.4 Performance Evaluation 107
7.4.1 Data Layout write overhead 107
7.4.2 Temporal Restore Search Performance 109
7.4.3 Spatial Restore Search Performance 111
7.5 Conclusions & Future Work 113

8 Conclusions & Future Directions 114

References 116

Appendix A. Acronyms 127

List of Tables

4.1 Storage ILM Chain Length 62
4.2 Backup Workload Statistics 67
6.1 Impact of Synchronous Naive File-Object CDP 85
6.2 Impact of Coalesce Factor (CF) 94
7.1 Impact of Data Layout on Journal Rewrites 108
7.2 Impact of Queueing on Restore spatial search 111
7.3 Impact of *Flush Interval* on Restore spatial search performance 113
A.1 Acronyms 127

List of Figures

1.1 Phases in lifecycle of data in an enterprise . . . 3
2.1 Breakdown of Power Usage in a Datacenter . . . 9
2.2 Disk States and Transitions . . . 10
2.3 Overview of Traditional Enterprise Backup Ecosystem . . . 21
2.4 Illustration of CDP logging with a sample timeline of operations . . . 23
2.5 Logical view of Swift Cloud Object Storage Cluster . . . 27
2.6 Physical view of Swift Cloud Object Storage Cluster . . . 28
3.1 GreenStor Storage Subsystem Architecture . . . 34
3.2 Hash Mapping Record for Extent-based Metadata Management . . . 38
3.3 Global Mapping Structure for Extent-based Metadata Management . . . 41
3.4 I/O Queueing Model for Delayed/Lazy Execution . . . 49
3.5 Impact of Scheduling Schemes on Percentage Energy Savings . . . 51
3.6 Impact of Scheduling Schemes on Average # of Disk Restarts . . . 53
3.7 Impact of Scheduling Schemes on Read Response Times . . . 54
3.8 Impact of cache/prefetch block allocation schemes on Metadata Efficiency 55
4.1 Data Collection from Traditional Enterprise backups . . . 59
4.2 Variation in Storage Configuration . . . 61
4.3 Variation in Client-Backup Server Associations . . . 63
4.4 Variation in Backup Window: Start Times . . . 64
4.5 Variation in Backup Window: Duration . . . 65
4.6 Breakdown of Backup Server Daily Ingest . . . 66
4.7 Variability across Backup Client Workloads . . . 68
5.1 Variation in Size of Backups . . . 72
5.2 Variation in Backup Client Throughput . . . 73

5.3 Backup Server Load Variability (Day1) 74
5.4 Backup Server Load Variability(Day2) 75
5.5 Adaptive Model based Backup Management Framework 76
5.6 Trace driven Enterprise Backup Simulation Framework 78
5.7 Modeling Accuracy for Backup Jobs 79
5.8 Improvement in Backup Performance with Model based Scheduling . . . 80
6.1 Naive File/Block Update to Object CDP 85
6.2 Performance Characteristics of Cloud Object Storage Cluster 87
6.3 cCDP System Overview 88
6.4 Object Structure: Journal & Data Associations 89
6.5 Impact of Coalesce Factor on Edge Buffer Usage 93
7.1 Retrieval Operations supported by Cloud Object Storage(Swift) 96
7.2 Retrieval Performance Characteristics of Object Storage (Swift): List vs Head vs Get 98
7.3 Naive Layout (1 Data Container, 1 Journal Container) 99
7.4 Btree based Temporally Indexed Layout (1 Data Container, 1 Journal Container) 100
7.5 Hybrid Layout (1 Data Container, n Journal Containers, 1 Index Container) 101
7.6 FIFO based Least Common Prefix 105
7.7 Trie based Least Common Prefix 107
7.8 Impact of Data Layout on PUT performance 107
7.9 Impact of Data Layout on Edge Buffer Overhead 109
7.10 Temporal Search Performance: Identification & Ordering Time 110
7.11 Comparison of FIFO and Trie LCP Queuing 111
7.12 Impact of *Flush Interval* on Restore spatial search performance 112

Chapter 1

Introduction

Recent trends of accelerated adoption of data driven decision making, Internet of Things (IoT), and in general, an explosion in richness of data have led to unprecedented changes in data storage requirements. From the typical paradigms of online, nearline and offline data, now the boundaries are increasingly blurred with more fuzzy interactions between real time transactions and analytics workloads.

From an infrastructure perspective, this exponential data growth introduces new challenges for cost optimization. Optimizing Total Cost of Ownership (TCO) of data storage infrastructure characterized by capital costs and operational management costs necessitate a radical redesign to cope with exponential data growth not just in terms of capacity, but also in the veracity and velocity of data. Economies of scale provided by warehouse style cloud computing and advances in material science leading to higher aerial densities provide new levers for capital cost optimization. Operational management costs, however, require re-design of data and systems management techniques.

An Overview of data lifecycle and the different management processes that interact with data during its lifecycle are shown in figure 1.1. Data is either created organically by applications or acquired by various means such as sensors. Enterprise data centers typically use a myriad of storage systems each with different cost, performance and power tradeoff. In general, these devices can be classified into Tiers of storage, with-

1. Tier 0 being the fastest performance (latencies in microseconds/sub millisecond range), high cost and high at rest and active power usage. SSD/flash based storage

typically falls in this category.

2. Tier 1 being slightly slower (latencies in a few milliseconds), slightly cheaper than SSDs and high at rest and active power usage. 15K RPM SAS storage typically falls in this category.

3. Tier 2 being slightly slower than Tier 1 (latencies in the 10-20 millisecond range), slightly cheaper than Tier 1 and moderate at rest and active power usage. 7500 RPM SATA storage typically falls in this category.

4. Nearline being much slower than Tier 2 (latencies in the 20-200 millisecond range), much cheaper than Tier 2 and moderate at rest and active power usage. Storage built using commodity non enterprise grade hardware falls in this category.

5. Archive being the slowest (latencies in minutes to get to first byte of data, with moderate streaming performance thereafter), extremely cheap with minimal to none at rest power consumption and very low active power usage. Tape based systems typically fall in this category.

One of the main aspects of cost effective data management in an enterprise is to constantly evaluate the business value of data and place it on the appropriate tier of storage for over cost optimization. At acquisition/creation, data may be placed on Tier 0/1 storage and over its lifetime, data may get moved across to lower more cost effective storage tiers. From a data protection perspective, data protection requirements vary over lifecycle of data. When primary accesses to data are prevalent, one needs to design protection mechanisms to protect against both logical and physical data corruption scenarios. Once the data moves to nearline or archival storage, the need for Point in Time (PiT) solutions such as backup and checkpoint reduces while the physical protection needs still exist.

1.1 Research Challenges

A significant body of research is focused on optimizing the power consumption of different media types and optimizing cost of data protection from a capacity optimization perspective. Compression and deduplication technologies have been heavily used in

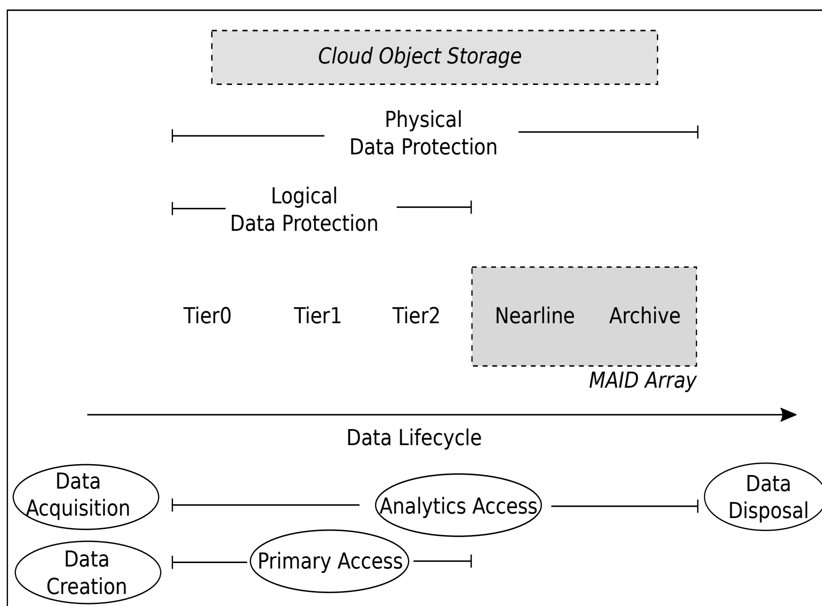


Figure 1.1: Phases in lifecycle of data in an enterprise

data protection systems to reduce data storage costs. Application access patterns have been heavily exploited for performance management. However, use of application access characteristics to continuously optimized underlying storage components for power usage is still relatively unexplored. Simple optimizations like recency of data access have been used in literature to dynamically move data between different storage devices with different power profile. These solutions mainly work in application with clear/distinct separation between online, nearline and offline data. Given the recent evolution of access patterns, combined real time and analytics workloads and newer media power management and data protection techniques, some of the new research challenges are-

1. *Evolution of access patterns and their impact on power management*: Typical enterprise application data access patterns have been studied extensively in multiple contexts - caching, tiering. Access patterns for database applications, file servers and other enterprise applications are well documented. With the advent of big data analytics often times on same data, nature of access patterns have changed significantly. Typical algorithms such as Least Recently Used (LRU) are no along applicable to these new types of data access. These applications employ

complex data replication mechanisms in software for resiliency and availability and are written to exploit the semantics of replication for performance. Research challenge is to capture such application semantics in conjunction with regular storage level access patterns and use for caching, power and other types of system optimizations.

2. *Exploiting access patterns to build power efficient storage platforms*: Typical hierarchical storage management policies are built around spatial and temporal access patterns that can easily be inferred at the storage layer by observing requests over a period of time. With evolving access patterns and application semantics, one research challenge is to provide an efficient mechanism for relaying such application hints from application layer to the storage layer. Standard storage protocols offer very limited support for such message passing. Given that most storage systems are inherently multi-tenant, with multiple applications, another research challenge is to design hint exploitation techniques that work across multiple application patterns. Ensuring fairness becomes more challenging than typical LRU based systems.

3. *Improving operational efficiencies of data protection systems*: Traditional data protection systems provide a myriad of choices which administrators can exploit to meet protection requirements. Management of these systems is through static policies defined by skilled domain experts. With economies of scale, rapid consolidation, static policy management can lead to significant inefficiencies. Typical administration tasks such as backup scheduling, problem diagnosis become extremely challenging leading to significant increase in operational costs. Research challenge is to tailor policies dynamically to different application requirements and performance characteristics of data protection systems in shared multi tenant systems.

4. *Exploiting software defined commodity storage for building enterprise data protection systems*: Scaling performance and capacity of data protection infrastructure to keep pace with scaling of production systems is a challenge. Specifically, given that cost efficiency requirements of data protection systems are much more stringent than production infrastructure, scaling data protection systems becomes

challenging. Research challenge is how to exploit newer trends such as software defined commodity storage systems for building cost effective, yet highly scalable and enterprise service level compliant data protection systems.

1.2 Scope & Contributions

Our work focusses on a) data management techniques for optimizing storage power usage and b) techniques for optimizing cost of data protection systems. In this section we outline the specific contributions of our work towards improving efficiency of data management.

1.2.1 Improving operational efficiency thru application aided storage power management

Enterprise environments and high performance computing are making use of large disk-based solutions that consume power all the time, unlike tape-based solutions in order to cope with changing application requirements. Consequently, the energy consumption of the storage solutions which contributes significantly to operational efficiency of data management has grown significantly. Our research is centered on the idea of using application hinting on top of Massive Array of Idle Disks (MAID) systems to build an energy efficient, near-online storage solution. The focus of our research is to identify and address the challenges in building a MAID-based storage solution that can exploit application hints efficiently. We propose a storage solution called GreenStor, which makes use of application hinting on top of MAID to improve energy efficiency. GreenStor is centered on MAID, with an architecture tailored towards more efficient data movement in order to aid in energy conservation. Specifically, we propose an extent-based metadata manager that achieves better space efficiency without sacrificing cache utilization and an opportunistic scheduling scheme that helps provide better use of application hints in a MAID system.

1.2.2 Improving operational efficiency and scalability of data protection through model based approaches

Data protection is one of the fundamental tasks in enterprise data management. Operational inefficiencies and scalability issues in data protection systems mainly stem from usage of static policy based management. We propose a model based dynamic backup scheduling framework that attempts to address key scalability and performance limitations of current backup systems. Specifically, we model backup performance as a function of both client and server side load characteristics along with configuration attributes and subsequently use these models for making smart backup scheduling decisions dynamically at runtime.

1.2.3 Improving cost efficiencies of data protection using commodity Software Defined Storage (SDS)

Continuous Data Protection (CDP) enables recoverability to any point in time (time travel) facilitated via journaling of every write made by a system to disk. Stringent storage performance and capacity requirements for journaling make CDP a very high cost solution leading to limited adoption. We propose cCDP - a Cloud Continuous Data Protection framework that efficiently combines cloud object stores with edge caching to address requirements of low cost, high capacity, low latency and high storage throughput. cCDP with careful tuning can not only meet but surpasses the write throughput and latency requirements of CDP with minimal buffer overheads.

1.2.4 Improving operational efficiency through smart restore optimization using commodity SDS

Data protection offered by CDP is significantly richer when compared to traditional data protection solutions. Operational efficiency and usability of CDP is however a function of how efficiently data can be restored in case of a failure. Specifically, ability to quickly search thru CDP logs stored on commodity SDS such as object stores to identify data that needs to be restored and subsequently fast tracking the restore data transfer operation are key to CDP. We propose a) a novel method of organizing layout of CDP logs on object storage to exploit object storage retrieval characteristics and

optimize temporal search performance, b) an object naming encoding scheme coupled with a Trie based queueing mechanism to optimize spatial search performance of CDP based restores.

1.3 Organization

Chapter 2 starts with background and related work for power management and data protection systems. Chapter 3 details use of application hinting on top of MAID systems to build an energy efficient, near-online storage solution. Chapter 4 highlights key insights gained from data characterization study of data protection systems from several production datacenters. Chapter 5 shows a model based backup scheduling system aimed at improving backup performance and predictability. Chapter 6 provides an overview of our proposed cloud object based continuous data protection system. Chapter 7 provides an overview of recovery optimizations in object based continuous data protection systems. Chapter 8 summaries the conclusions and provides an overview of future directions.

Chapter 2

Background & Related Work

Understanding the nature of power usage in a data center is the key first step in any approach to improve power efficiency. In this chapter we first explore power consumption characteristics of data centers, followed by a deep dive into storage power management. We conclude with research challenges in data management for power optimization.

2.1 Datacenter Power Usage

Power consumption in data centers can be broadly attributed to two classes of equipment, namely - Heating, Ventilation & Air Conditioning (HVAC) equipment and Information Technology (IT) equipment. Environment Protection Agency(EPA) report on server and data center energy efficiency provides a good breakdown on data center energy consumption [1].

A typical data center houses hundreds or even thousands of servers and storage equipment. All IT equipment that consumes power generates heat. It is the job of HVAC equipment to extract/capture this generated heat and cool it down so as to maintain appropriate operating temperature in the data center. HVAC equipment that is needed to maintain appropriate ambient conditions, in general accounts for nearly 50% of the overall data center power consumption. In general, for every watt of compute/storage power consumed, HVAC can take anywhere between 0.5 to 2W to cool down the infrastructure.

IT equipment account for the remaining 50% of consumed power in a data center.

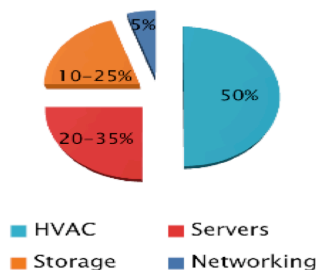


Figure 2.1: Breakdown of Power Usage in a Datacenter

IT equipment range from commodity servers to highly sophisticated blade servers and supercomputers, from simple direct attached storage to complex clusters of high end disk arrays, from simple ethernet based network equipment to high end Storage Area Network (SAN) equipment. The breakdown of power consumption amongst IT equipment largely depends on the type of data center in question. Compute centric data centers typically run applications that consume a lot of CPU resources and do not necessarily store much data. Certain type of HPC data centers are of this category. In such data centers, servers account for about 70% of IT power budget, where as storage and networking equipment account for about 20% and 10% respectively. Data centric data centers typically run applications that do a mix of computation and data storage. Enterprise data centers are typically of this category. Emphasis of enterprise data centers is not just on computation, but also on efficient storage and backup of large volumes of processed data either for further processing or due to retention requirements of applications. In such data centers, servers account for about 40% of IT power budget, where as storage and network equipment account for about 50% and 10% respectively. In general, power budgets of IT equipment in a data center can be summarized as - 40 - 70% for servers, 20 - 50% for storage and about 10% for networking equipment of the overall IT power budget.

2.2 Storage Power Usage

Data storage in data centers vary widely from simple direct attached storage to complex network attached storage solutions. Enterprise data centers typically tend to have more

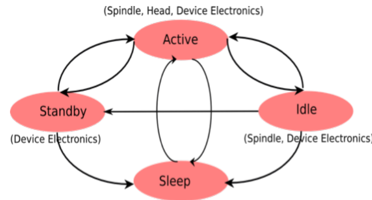


Figure 2.2: Disk States and Transitions

network attached storage solutions due to the stringent requirements of availability and reliability. In addition, scalability is a major concern for enterprise systems. Typical network storage systems range from a few tens of disk drives to several hundred or thousand disk drives. Systems of this kind which are used for online or primary storage tier typically use high-end SCSI/SATA drives operating at speeds of 10-15K RPM. Power consumption of these magnetic disks is function of its rotational speed and the data access rate. Majority of power is consumed by the rotating spindle, followed by the head assembly that moves along the platters to requested sectors/Logical Block Address(LBAs) and the buffers used for queuing requests and requested data.

A mechanical hard disk typically has four different operational states, namely -

- Active: The head assembly and the buffer are both *On* and the spindle is rotating at its full speed. In this state, power consumption varies based on head movement. If read/write requests are sequential in nature, head movement is relatively minimal and hence power consumption is mainly determined by the spindle's rotational speed.

- Idle: The head assembly and the buffer are both *On* and the spindle is rotating at its full speed. However, in this state no requests are actually being serviced. Power consumed is mainly determined by the spindle's rotational speed. Transiting between *Idle* and *Active* states is instantaneous.

- Standby/Sleep: The head assembly is *Off* and the spindle is at rest, but the buffers are *On* which facilitates queuing of new requests and the disk is still able to respond to diagnostic queries by the controlling system. Power consumed in

this state is very little compared to *Active* or *Idle* state. The main disadvantage, however, is that transitioning from *Standby* to *Idle* or *Active* takes a long time, typically about 8-10 seconds to get the spindle running at its specified speed.

- Off: All three components are *Off*. The disk cannot accept commands for queuing and cannot respond to diagnostic queries.

The power consumed by the spindle motor ($P_{spindle}$) is directly proportional to the square of its angular velocity (ω).

$$P_{spindle} \; \alpha \; \omega^2 \tag{2.1}$$

For a typical SCSI disk rotating at 15K rpm, this translates to about 5-6W. The magnitude of head movement is dictated by the access pattern. On an average, the head assembly accounts for about 2-4W of power consumption. In summary, *Active* state power consumption is about 8-11W, where as in *Idle* state it is about 6-7W and about 1W in *Standby* state. In addition, when placed in large disk array based systems, the array controllers and enclosures account for 1-2W of power per disk. In comparison with their server counterparts, in disks, utilization based variation is relatively lesser, i.e., idle mode power usage is more significant for disks.

Though storage devices consume a lot less power when compared to their server counterparts, In enterprise storage solutions that made up of hundreds or even thousands of disks, storage power consumption is a significant cause for concern.

2.3 Taxonomy of Storage Power Optimization Techniques

Storage power management has been an active area of research in the context of hyper scale systems. In this section we provide a taxonomy of different sub topics explored by researchers in domain of storage power management.

2.3.1 Adaptive spin down & Dynamic modulation of rotation speed

Given that spindle motor consumes most of the disk power, these techniques try to turn off the spindle motor (sleep mode) by monitoring workload idle periods. Most research in this area revolves around idle period prediction and minimizing the impact of these

transitions on disk responsiveness as transition time is usually around 8 to 10s. Considering the variation of power usage with rotational speed of the spindle motor, these techniques try to build disks that can adaptive transition between different rotational speeds based on workload characteristics. Dynamic rotation control or dynamically changing disk rotation speed is an idea that has been around for a while. Disk manufacturers today produce two-speed disks, with the restriction that two-speed disks cannot serve requests in low speed, they have to be transitioned back to high speed first.

Gurumurthi *et al.* investigated a hypothetical case where a disk can change its rotational speeds on the fly [2,3] . Using this assumption, the authors develop a policy for optimizing power consumption. Accordingly, based the response time of disks, the rotational speed of the disk is altered. A fast response time that is greater than specified or expected threshold is a waste of performance. The idea here is to limit this wastage of performance by switching the rotational velocity of the disk to a lower value that still yields acceptable performance. In the process a power model for multi speed disks is developed and the practical limitation of this approach is discussed. The main hurdle for this mechanism is that the feasibility of developing a single disk that can change speeds on the fly in a cost effective manner is very unlikely and hence the practicality of the idea reduces. Some simulation was conducted for synthetic workloads and it shows the proposed scheme can yield a power savings of up to 60%.

Different disk types have different power usage characteristics. Carrera *et al.* evaluated various alternatives for conserving power in enterprise disks [4] [5]. Four different alternatives were compared. These include powering down during idle periods (leveraging work on laptop drives), replacing high performance SCSI disks with a set of lower power disks that can provide same performance, and reliability, combining SCSI and laptop disks such that only one is *Active/On* at any time and multi-speed disks. For the first two approaches, the authors used analytical models to show that a solution using these two approaches is infeasible in terms of power savings. For combined SCSI and multi-speed disks the authors resort to emulation due to lack of availability of such hardware. These emulations show that only multi-speed disks (10K and 15K together) can provide energy savings of up to 23% without sacrificing any performance for regular network server workloads. Overall, the study concludes that although current manufacturing techniques do not facilitate production of such multi-speed disks, if energy

conservation in enterprise systems is to be achieved, disk manufacturers need to consider the benefits carefully and try to push their design to overcome this limit.

2.3.2 Optimizing Data Layouts

Strictly controlling disk accesses by optimizing data layouts is one way of skewing disk access pattern, i.e., changing disk idle periods. Pinheiro *et al.* proposed the idea of making use of skewed file access frequencies in order to optimize power consumption of disk array based servers [5]. The technique called Popular Data Concentration (PDC) works by classifying the data based on file popularity and then migrate the most popular files to a subset of disks thereby increasing the idle periods of remaining disks. Maximizing idle time helps in making more transitions to sleep state and hence more power can be conserved. Limitation of this approach is that access of unpopular files could potentially involve turning *On* or spinning up a disk, which could take about 8-12 seconds typically before actual data access can be made.

Hibernator is a disk array design proposed by Zhu *et al.* for optimizing storage power consumption [6]. This work again assumes availability of multi-speed disks and tries to dynamically create and maintain multiple tiers of disks, each at a different rotational speed. Performance is used as a feedback in order to adjust the number of disks in each tier and the speed of the disks themselves. A disk speed determination algorithm and efficient mechanisms for exchange of data between various tiers were developed. Based on trace driven simulations, the authors showed an energy savings of about 65% for file system based workloads. Further, the authors benchmarked an emulated system with DB2 transaction processing engine and showed an energy savings of about 29%.

Massive Array of Idle Disks (MAID) is a technique proposed by Colarelli *et al.* which again involves data migration [7] . Unlike PDC, MAID tries to copy files based on their temporal locality. MAID uses a small subset of disks as dedicated cache disks and uses traditional methods to exploit temporal locality. The remaining disks are turned on on-demand. Again, this scheme too suffers from the fact that files that have not been accessed in recent past could potentially have a retrieval time in tens of seconds. Due to this high performance penalty, MAID is more suited to near-line or tertiary storage environments such as disk based archival or backup solutions. COPAN systems is a startup which has built a successfully archival product based on the idea. Their

archival systems claim to outperform traditional archival and backup products both in terms of performance and power consumption.

2.3.3 Caching for Power Optimization

Enterprise storage solutions typically have large amounts of cache in front of regular disks. These techniques make use of the cache to aid in disk power management. Specifically, research in this area focuses on cache management algorithms that are designed to minimize disk power usage, either by minimizing disk access or by increasing length of idle periods. Disk arrays generally have large caches in order to speed up access for both read and write requests. One could in effect use these huge caches to increase idle periods of disks and in doing so can help more disks to transition to sleep state thereby improving energy efficiency. Zhu *et al.* consider various cache management algorithms centered around this idea [8] , [9]. The authors proposed two new cache management algorithms, - Partition Aware LRU (PALRU) and Partition Based LRU (PBLRU). PALRU classifies all disks based on access patterns into two classes - Priority (disks with fewer cold misses and longer idle times) and Regular and maintains two separate LRU queues. At the time of an eviction decision, first the regular queue elements are chosen as victims. If regular queue is empty, the algorithm chooses elements from priority queue as victims. PBLRU on the other hand, differentiates between disks by dynamically varying the number of allocated cache blocks per disk. It divides the cache into multiple partitions, one per disk and adjusts the size of these partitions periodically based on workload characteristics. The simulation results with OLTP traces show that PALRU consumes 14-16% less energy and PBLRU consumes 11-13% less energy than traditional LRU. On the other hand, for the Cello96 trace (file system trace), PALRU saves less than 1% energy over LRU while PBLRU is 7.6-7.7% more energy-efficient than LRU. The authors attribute this decimal performance improvement for file system trace to nearly 64% cold misses in the trace used. For write requests, they compared traditional Write Back (WB) and Write Through (WT) policies with respect to energy management and find that with a read/write ratio of 50%, the energy savings of WB compared to WT is about 10%. Based on insights from this comparison, the authors proposed Write back with eager updates (WBEU) and Write Through with Deferred

Updates (WTDU) which can yield about 65% and 55% more energy savings than traditional WT when 100% of disk access are writes respectively.

Since write requests in enterprise storage devices almost never directly gets written to target disks (cached instead), Narayanan *et al.* investigated the use of write offloading as mechanism to conserve disk power usage [10]. Specifically, the authors examined block level traces of a typical mid size enterprise comprising of 36 volumes. Of these 36 volumes, they found that about 19 volumes are write dominated with a read/write ratio of about 0.18. For these volumes, they also found that significant idle periods exist and idle periods increase substantially if writes are not considered. Based on this observation, a system designed for write offloading for write dominated volumes was proposed. Write offloading facilitates complete spin downs of volumes periodically thereby aiding in significant power savings. Performance evaluation of the prototype with replayed traces show that on average about 45-60% energy savings can be achieved for these write dominated volumes by using write offloading.

2.3.4 RAID Adaptations

Redundancy is a key feature of most enterprise storage solution. Mechanisms like Redundant Array of Inexpensive Disks (RAID) play a key role in providing different levels of redundancy. In such systems, another new degree of freedom, that of scheduling redundancy related operations can be exploited for power management purposes. One such idea is EERAID developed by Wang *et al.* [11]. EERAID is a RAID engine aimed at minimizing energy consumption of RAID disks by adaptively scheduling requests to various disks that form the RAID group. Specifically, by controlling the mapping of logical request to a RAID stripe, disk idle period of a sub set of disks is maximized facilitating spin down of these disks. For RAID 1, the authors proposed Windowed Round Robin scheduler that dispatches a window of requests to one RAID disk before switching to the other RAID disks and vice versa. For RAID 5, the authors proposed Transformable Reads. The main idea is that for a read request of a stripe that is currently on spun down disk, the stripe is reconstructed using other data blocks and parity blocks that for the stripe. Further, the authors proposed power aware de-stage algorithm to accommodate write requests in the design of EERAID.

Weddle *et al.* proposed Power Aware RAID (PARAID) in [12]. With the observation

that system load usually varies quite a bit depending on time of day, PARAID tries to dynamically vary the number of powered on disks to satisfy this varying load. The authors used the analogy of gear shifting in vehicles to draw a parallel in server loads. In addition, to tackle the problem of high penalty due to requests for data on spun down disks, PARAID maintains a skewed data layout. Specifically, free space on *On* disks is used to store redundant copies of data present on spun down disks. A gear is characterized by number of *On* disks and a gear up shift amounts to increasing number of powered on disks to cope with increased performance demand and similarly a gear downshift amounts to spinning down additional drives in response to reduction in system load. The prototype built on Linux software RAID driver shows a power savings of about 34% as compared to power unaware RAID 5. An interesting design consideration explored by PARIAD is an approximate method of controlling reliability by limiting number of spin ups and spin downs of disks.

2.3.5 Emerging Media Types

In recent times, Flash and solid state media have gained increased traction due to their performance potential and higher power efficiencies. Research in this area focuses on integrating flash/solid state disks into the storage hierarchy. Specifically, by using flash/solid state disk tier just above the magnetic disk tier as a layer of large cache, magnetic disk idle periods can be extended and accesses to disk can be optimized. NAND-based flash memory is a non-volatile data storage device. It has rapidly increased in popularity as the primary data storage medium for mobile devices, such as phones, digital cameras, and sensor devices. This type device is popular due to its small size, low weight, low power consumption, high shock resistance, and fast read performance [13] [14]. Flash memory has also started to penetrate the markets from laptops, PCs, to enterprise-class server domains [15] [16] [17] [18]. For examples, companies like Dell and Samsung have already launched laptops with only flash memory based Solid State Drives (SSDs) [19]. Samsung, STec and SimpleTech have launched SSDs with better performance compared to the traditional 15000 RPM enterprise disk drives [20] [21]. Enterprise class SSDs provide unique opportunities to boost up the I/O-intensive applications performance in the data centers [18]. Compared to hard disks, flash based SSDs

are very attractive in the high-end servers of data centers due to their faster read performance, lower cooling cost, and higher power savings. Data in SSD is read and write by pages. The read for a page in SSD is about 20 times fast than that of a page read from hard disks. Unlike hard disks, there are no differences in terms of time between random reads and sequential reads. It is best suited for caching data between memory and hard disks since the hard disks still hold advantage of cost and storage capacity. However, relatively low write performance and longevity problem of flash memory require new and innovative solutions to fully incorporate flash based SSDs into the high-end servers of data centers. Unlike the conventional magnetic disks, where read and write operations exhibit symmetric speed, in flash based SSDs, the write operation is substantially slower than the read operation. This asymmetry arises as flash memory does not allow overwrite and write operations in a flash memory must be preceded by an erase operation. Typically a block spans 64-128 pages and live pages from a block need to be moved to new pages before the erase is being done. Furthermore, a flash memory block can only be erased for a limited number of times, after which it acts like a read-only device [14]. These slow write performance and wear-out issues are the two most important concerns restricting SSDs wide spread acceptance in the data centers [9]. Existing approaches to overcome these problems through modified Flash Translation Layer (FTL) is effective for sequential write patterns, however random write patterns are still very slow [14].

Flash memory has the lowest power consumption rate of active devices when compared with both DRAM and hard disks. For example, the power consumption for a 128GB SSD is about 2w, that of a DRAM DIMM Module of 1GB is 5w, and 17.5w (12.6w) for a 15,000 RPM 300GB (7,200 RPM 750GB) hard drive. If a flash memory based device like SSD can be used for cashing most frequently accessed files and the first portion of all the other files, the hard disks can be put into idle mode most of the time to save energy. The most frequent accessed data will be accessed from SSD. This will not only provide better energy saving, but also faster access time. When a file stored on hard disk is accessed, the first portion of the data can be accessed from SSD. This will buy some time for the idle hard disk to turn into active mode. Innovated research using flash memory based storage devices in enterprise-class storage solutions that address energy saving, performance, and wear-out issues together are still in high demand.

Application hinting has been studied in depth by several researchers. [22], [23], and [24] deal with the use of application hinting to minimize application response times by actively prefetching hinted data from disks into main memory ahead of time. Specifically, they explore tradeoffs of various scheduling schemes built using cost-benefit analysis. [25] explores the idea of cooperative processes to achieve energy conservation. Specifically, the authors propose a system where applications/processes provide hints (about IO operations) to the operating systems to aid in energy conservation of all types of IO devices. The main difference in our work, is that we focus on consolidated storage systems that are shared across multiple servers.

To the best of our knowledge, the idea of application hinting has not been explored before in the context of storage system energy conservation, specifically in the context of large shared disk farms. Our work presented in Chapter 3 primarily deals with taking an existing design (i.e., MAID) and optimizing the design for large-scale nearline storage using application hinting. We do not deal with multi-speed or DRPM disks. Instead, we assume standard disks in a MAID configuration with the ability to intelligently transition between Standby and ON states.

2.4 Data Protection

Understanding the paradigms of data protection systems in a data center is the key first step in any approach to optimize cost efficiencies of data protection. Based on nature and type of application in question and its business value, enterprises typically use a combination of one of more data protection strategies to safeguard data. Data corruption can be classified into two categories, namely -

1. *Logical corruption*: Application errors, software bugs, malicious virus attacks and user generated errors can often cause significant data corruption. In order to recover from such corruption scenarios, data needs to be reverted back to a prior known healthy point in time and subsequent operations need to be selectively reapplied under close supervision. Having multiple synchronous replication of data does not help in such scenarios as errors propagate to the replicas as well.

2. *Physical corruption*: Hardware errors, silent disk errors such as disk bit flips, partial or total system failures cause a different type of corruption. In order to

recover from such corruption scenarios, a different synchronous copy of data needs to be promoted and all applications accessing data need to be redirected to this new replica.

Recall from figure 1.1, at different points in lifecycle of data, data protection requirements change. For instance, when data is relatively new/fresh, it typically gets placed on higher performing tiers of storage and during this period, both logical and physical data protection strategies are required to safeguard data. Over time as data access reduces, and data gets migrated to lower performing tiers, need for logical protection reduces.

2.5 Taxonomy of Data Protection Technologies

Data Protection techniques used by enterprises to safeguard against different types of corruption can be categorized into two main categories, namely - Point in Time(PiT) and Mirroring. Focus of Point in Time data protection technologies is to provide a conceptual equivalent of a checkpoint so that data can be rolled back or rolled forward for purposes of recovery. These techniques can further be classified into three groups as follows,

2.5.1 Appliance based Backup

Traditional backups have been the defacto data protection technology in most enterprises for several decades. Typically implemented as a client-server model, where in backup client software is installed on systems that need data protection and are configured to backup to one or more central backup server. Figure 2.3 shows overview of typical enterprise backup process. Different types of backup are typically supported such as - full, incremental and differential. Policies govern the type and scheduling of backups. From a recovery perspective, this class of technologies can help recover from logical and physical corruption scenarios. However, potential for data loss or unrecoverable data is a function of the scheduling policies in place. Further, time to recover is significant compared to other techniques as full copy operations are involved along with rollforward/rollback of data.

Methodology of backups has remained relatively unchanged in recent decades. Multiple enterprise servers (hosting apps, databases, files,etc..) backup to one or more backup servers based on policies that govern scheduling, retention and lifecycle processes. Process starts by installing backup agent software on client server(s)/hypervisors followed by a one time configuration operation of selecting appropriate backup server and policies of backup process. Ones this initial configuration is completed, the agent software based on defined policies coordinates with corresponding backup server to determine what needs to be backed up and initiates a copy of the identified files or disk images (F in figure 2.3) to the backup server. Copy process could either be over the regular or dedicated networks or could be Local Area Network (LAN) free via Storage Area Networks (SAN). On receiving files/images from client(s), backup server applies a variety of transformations based on defined policies (compaction - merge small files into larger containers,metadata extraction,capacity optimization techniques such as compression,deduplication,etc..) and writes these transformed streams (S in figure 2.3) to storage devices defined by policies and updates its catalogs.

Apart from this inline client interaction, backup servers perform a range of asynchronous background tasks to enforce policies, such as tiering, retention enforcement, deduplication,etc.. Specifically, most backup servers implement Hierarchical Storage management(HSM) or tiering for balancing cost and performance. An example policies would be to keep backups in disk pool until capacity utilization reaches a high watermark and then tier or migrate it to tape pool. Additionally, some backup servers, perform asynchronous deduplication/compression as a background process in order to limit impact on inline client data ingestion. Alternatively, backup servers do offload capacity optimization to backend storage appliances which offer such functionality. Optionally, some backup servers mirror/replicate their data across storage pools for enhanced reliability. These replica pools are typically on different failure domains and often differ in media types. In practice, defining policies is more of an art than a science and is typically handled by skilled domain experts at provisioning time and is rarely revisited. Selection a backup server, and storage device/pools is mainly done based on rules of thumb or tenancy requirements. Specifying a schedule/backup window for backup operations is done based on application owners domain knowledge of system load. Two main

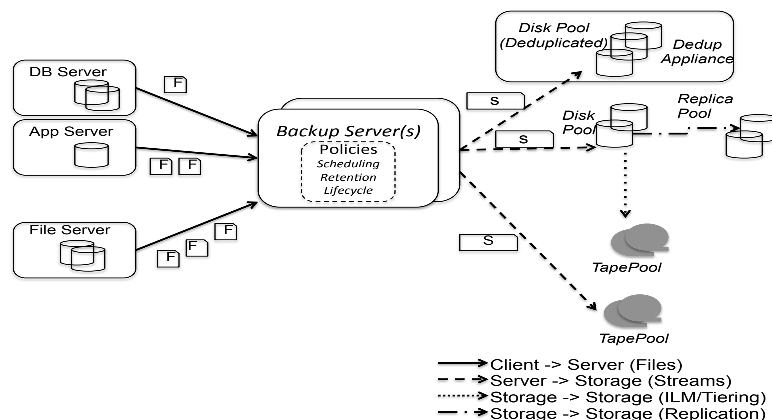


Figure 2.3: Overview of Traditional Enterprise Backup Ecosystem

flavors of scheduling typically used are - a) backup server initiated within a client pre-specified backup window, b) client initiated - triggered by either cron type schedulers or by applications/users on-demand. In the former case, backup server determines when to trigger backup operation based on its internal resource availability within the window. On any given backup server, it is common to find a mix of both types of schedules. These static policies are usually configured by domain experts at initial provisioning time and are seldom revisited.

Schedules vary widely based on - type of data being protected (business value) and type of data protection system in place. For instance, business critical applications typically employ schedules that backup multiple times a day. Default schedule typically is to backup once daily. Further, data protection systems influence schedule as well. Specifically, two categories of systems exists - uniform and cyclic. Uniform systems typically backup all data ones at the beginning (full backup) and employ incremental backups there on. Cyclic systems on the other hand resort to weekend full backups followed by weekday incremental.

Backup systems have been a very well explored area of research. In recent times, focus of most of research has been on capacity optimization and specifically on deduplication and compression related technologies [26] [27] [28] [29]. Quinlan et al. [30] were one of the first to propose deep integration of deduplication into archival systems. [31] proposed an optimized delta compression scheme to aid in Wide Area Network (WAN)

scale replication of backup datasets. [32] explored tradeoff in design of routing systems for deduplication. [33] propose new technique for inline deduplication using locality aware sampling. Zhu et al. [34] focuses on solving disk bandwidth issues in large scale deduplication systems. Our work explores backup server performance optimization via optimized scheduling of backups which is complementary to above reviewed literature.

Wallace et al. [35] studied characteristics of backup systems. Their study compares backup storage systems with primary storage ones. Specifically, they compare file characteristics of primary storage systems to stream characteristics seen on deduplication appliances. Using this characterization of backup systems they explore implications on caching for deduplication techniques. In our earlier work [36] , we explored space efficiencies and performance tradeoff of deduplication based backup systems. Birke et al. [37] characterized the velocity and variety of storage data collected across a large set of thousands of virtual machines. The study also analyzed impact of virtual machine colocation and consolidation on storage. On filesystem front, extensive work has focussed on volume and variety [38] [39] [40] [41]. General server workload characterization for purposes of evaluating energy efficiency was explored in [42]. [43] analyzes capacity requirements on backup storage appliances and proposes approaches to forecast growth.Our data characterization presented in Chapter 4 complements above mentioned literature and differs in the fact that we attempt to characterize the backup process and backup servers from a configuration variety and load perspective.

2.5.2 Continuous Data Protection(CDP)

CDP enables recoverability to any point in time (time travel) facilitated via journaling of every write made by a system to disk. Unlike traditional backups which provide limited number of points in time based on policies in place, CDP provides near infinite points in time for rollback/roll-forward. Unlike backups which run typically one or two times a day based on predefined schedule policies, CDP sits in the data path and intercepts all IO operations for logging. *CDP* works by recording every change to the primary data [44]. CDP is commonly implemented in either the block [45] or the filesystem layer [46,47]. The filesystem layer provides additional semantics of data such as directory hierarchies which can be useful for optimizing CDP and providing more usable restore operations. In contract, block based CDP is easier to implement as the

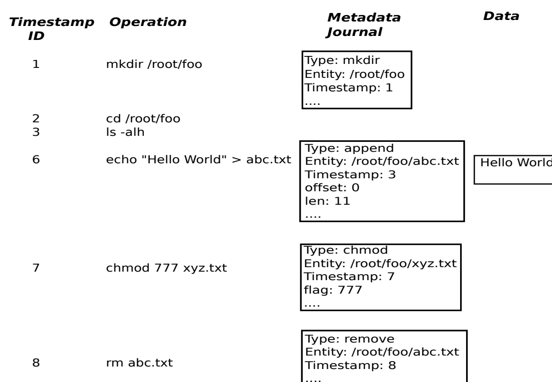


Figure 2.4: Illustration of CDP logging with a sample timeline of operations

same block driver can support a variety of filesystems on top. In this work we focus on filesystem level CDP. Figure 2.4 shows a sequence of operations at a filesystem layer and their corresponding CDP artifacts. Operations observed at a filesystem layer can be characterized into read operations on data and metadata (ID 2,3) which do not modify the system persistent state and hence do not require logging and write operations on data and metadata such as file create, directory remove, file write at offset (ID 1,6-8) that need to be logged to facilitate recovery. CDP intercepts only these write operations on data and metadata, timestamps them and records them in its *metadata journal* and *data journals* synchronously in primary data path with transactional semantics. CDP over the years has drawn significant interest both in academia and industry. Several commercial offerings implement CDP in different forms [46, 48, 49]. Majority of research in CDP deals various aspects of designing block based CDP systems.

Architecture of block based CDP has been studied in various contexts. Laden et al. [50] explore different designs of storage controller based CDP architectures and propose a model for predicting CDP overheads as a function of the workload on any given controller based CDP architecture. TRAP-Array [51] is a disk array based CDP implementation that focusses on timely recovery and space overheads of maintaining CDP data. Piggybacking on Exclusive-OR operations that are typically performed in RAID based controllers, TRAP-Array aims to provide CDP function with minimal added write performance overhead. Peabody [52] proposes block based implementation of CDP using network iSCSI targets. By exploiting content based coalescing, Peabody implements

an efficient block based network storage volume with significantly lesser write overheads and capacity savings. In contrast our work focusses on am alternate architecture of CDP with specific focus on Cloud object storage as the backend datastore.

Recovery and designing CDP for Point-In-time access is another area of CDP research. ST-CDP [44] extends on TRAP-Array based design with main focus on snapshotting. Using storage system snapshots, the authors propose a way of encoding snapshot information into parity logs maintained by TRAP-Array in order to reduce length of rollback or rollfoward internals. Further, a mathematical model is proposed to help in determining the frequency of snapshotting as a function of recovery time and space overhead. TH-CDP [53] is a block based CDP system that focusses on transparent checkpointing and virtual recovery images. For write overhead minimization aging a log structured approach is utilized. Using page prefetching, the authors show that read speeds for virtual history volumes can be significantly improved. Our work is an initial design of CDP with the new cloud object storage model. Integrating other replication technologies with CDP is key to CDP lifetime. We intend to explore these techniques in future work.

Efficiency of journal indexing is key to minimizing performance overheads and CDP lifecycle management. Mariner [54] is another iSCSI block based CDP system which focusses on use of track based logging to cater to both long and short term CDP logging requirements. Using a modified log structured filesystem for storing incoming writes, Mariner can cope with heavy write workloads with minimal performance overheads. Use of this write anywhere layout is a very effective technique to reduce CDP mirroring penalty. Lu et al. [55] focus on optimizing index updates in block based CDP systems. Authors show that index updates can easily become the prime bottleneck in scaling a CDP system. By using a combination of batching updates and committing index updates as larger sequential IO on log structured file layout, the authors show that performance overheads in index updates can be drastically reduced from 95% to under 15% of write traffic. Soules et al. [56] explore efficiency of metadata management in versioning filesystems. Authors propose a journal based metadata encoding for file history metadata logging and a multi-version Btree based index for directory history metadata logging. By extending CVFS with these efficient metadata management schemes, the authors show that associated space requirement for file logging can be reduced by 80%

and for directory logging by almost 99%. Object storage introduces significant new challenges in designing CDP logging for efficiency. Our work focusses on optimizing CDP logging using object storage backends.

Layout/organization of data in large scale file systems is an active area of research. Research in this domain ranges from static data placement strategies [57] to dynamic runtime optimization of data layouts for different workload access patterns [58,59]. Chiu et al. [60] focus on improving IO performance of parallel indexing via optimized Index file structures. Our work complements existing research in that we focus on object data layouts for storing CDP logs with temporal access patterns.

Zheng et al. [61] focus on addressing impedance mismatch between in memory databases and disk based logging/CDP required for durability. Using careful engineering optimizations, the authors show that durability and recoverability of in memory databases backed by on disk logging can be very efficiently implemented with minimal overheads. Our work is complementary and in that we focus on addressing impedance mismatch between block and cloud object storage systems.

Use of CDP and versioning for Intrusion detection and for protecting data in compromised systems has been extensively explored in Self-* storage systems by [62]. Use of CDP in combination with live migration checkpointing to enable virtual machine time travel is explored by [63].

In summary, our work in Chapters 6, 7 explore performance and usability issues with CDP when implemented over newer cloud object based storage architectures. Given the uniqueness of object stores and their differences from traditional and log structured filesystems employed by most of the above referenced literature, our work focusses on exploring the write performance and index management issues that arise from these differences.

2.5.3 Snapshotting & Mirroring

Several layers in the system stack provide functions to achieve checkpointing. Most enterprise storage controllers provide different types of snapshotting. Virtual machine management systems provide snapshotting support. Some enterprise applications themselves provide built-in snapshotting support. Most of these solutions provide protection

against logical corruption. However, protection against physical failures requires combining with other techniques.

Focus of mirroring technologies is to provide resiliency and high availability by replicating every IO to at least two different physical systems such that incase of a physical failure, the secondary replica can be instantly used for several application requests. These techniques can be further grouped based on synchronicity of operations.

1. Synchronous mirroring techniques sit in the data path and synchronously replicate every IO to a secondary physical system and applications are only acknowledged upon successfully commit on both systems.

2. Asynchronous mirroring techniques batch IO updates and apply to a secondary system in batches there by not impacting the primary latencies. However, these techniques are prone to a limited amount of data loss in case of failure.

Further, these techniques can be deployed to safeguard against either local system failures such a storage system or a more broader failure such as an entire datacenter.

2.6 Relative Efficiencies of Data Protection

Data protection costs are composed of two types of costs - operational management costs and capital/acquisition costs. Each of the data protection techniques outlined in previous section have different operational and capital cost considerations. Point in time techniques such as appliance based backups are the cheapest with regards to capital costs as most often they use cheaper storage technologies such as Tape for longterm storage leading to low initial acquisition and low ongoing power consumption costs. However, operational costs involved in day to day management are significantly higher due to increased complexity of policy management and increased complexity of restores.

CDP on the other hand, has very low operational costs as management complexity if minimal. However, the cost of acquisition is significantly higher owing to not just increased capacity requirements, but also increased throughput requirements.

Mirroring technologies have upfront cost of acquisition. Operational costs are lower due to reduced complexity of operations. However, such technologies cannot provide

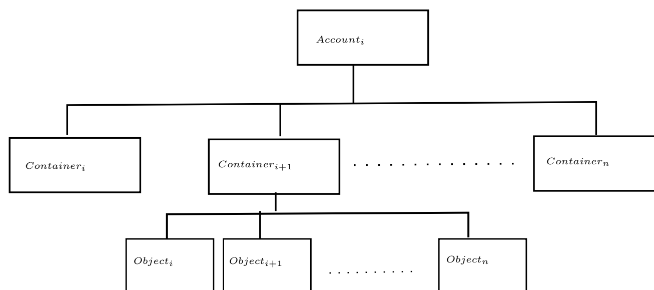


Figure 2.5: Logical view of Swift Cloud Object Storage Cluster

protection against all types of corruption and often require to be supplemented with other point in time technologies.

2.7 Software Defined Storage & Cost Efficiencies

Cloud Object Storage frameworks such as Openstack Swift [64], Amazon S3 [65] and Google Object store [66] expose an object abstraction supporting variable sized data and variable number of metadata keys to accompany the data. Figure 2.5 shows logical abstractions in Swift object storage. Unlike traditional filesystem that are designed to provide hierarchical grouping of files in directories and folders, Swift (and S3/Google Object store) provides a simple two level hierarchy - a top level for accounts and each account can further have variable number of containers (buckets being the logical equivalents in S3/Google Object store), and at the lower container level, within each container, one have any number of objects. Account abstraction is useful for enforcing tenancy requirements and containers can be used for grouping objects based on similar management policies or custom application data modeling.

Different operations on these logical abstractions are typically supported via HTTP get/put/post/delete REST based interfaces. Object access APIs have different access semantics as oppose to their block or filesystem counterparts, namely -

- Granularity of writes is a full object. No partial update operations are supported. Updating a byte of data within a large object will require full rewrite/PUT of the modified object.

- Read operations can either access the full object data or may choose to read

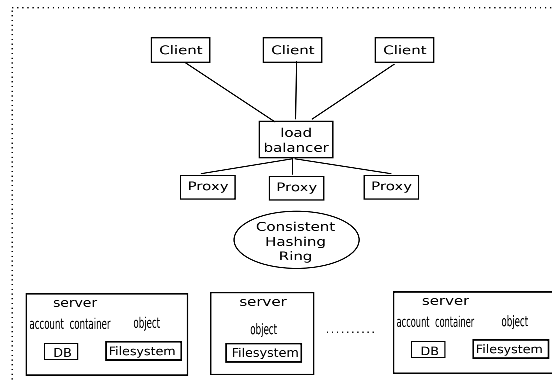


Figure 2.6: Physical view of Swift Cloud Object Storage Cluster

certain sections of the object data.

- List operations can become quite expensive as the number of objects in a container increases. Pagination is typically used to limit/batch the number of responses per list operation.
- Unlike block storage systems that provide a continuous address space via Logical Block Addressing (LBAs), the onus is on the application/writer to achieve similar semantics via object name encoding or object metadata.

Using such simplified access interfaces hides application developers from the hassle of filesystem/volume management. From an infrastructure provider/maintainer's perspective, this transparent object API facilitates seamless horizontal scaling and performance management, oblivious to application users. Figure 2.6 provides a overview of physical deployment of swift object storage. Most of these object implementations are built on top of commodity hardware with the goal of minimizing cost. The micro services shown are hardware agnostic and can be deployed on a variety of infrastructure components. On the performance front, analogous to block size, object size has a similar impact on performance of read and write operations. However, the clear dichotomy of random and sequential operations does not apply in case of object storage. In summary, software defined storage systems such as object stores provide significant flexibility in deployment which one can use to optimize for cost, scalability, availability or all of the above.

Chapter 3

Application-Aided Energy-Efficient Storage

To cope with unprecedented growth data, high performance computing and enterprise environments are making use of large disk-based solutions that consume power all the time, unlike tape-based solutions. Consequently, the energy consumption of the storage solutions which contributes significantly to operational efficiency of data management has grown significantly. In this chapter we propose a storage solution called GreenStor, which makes use of application hinting on top of Massive Arrays of Idle Disks (MAID) to improve energy efficiency. GreenStor is centered on MAID, but with more efficient data movement to aid in energy conservation. Specifically, we propose an extent-based meta-data manager that achieves better space efficiency without sacrificing cache utilization and an opportunistic scheduling scheme that helps provide better use of application hints in a MAID system. Preliminary results show that our proposed opportunistic scheme for application hint scheduling consumes up to 40% less energy compared to traditional non-MAID storage solutions, whereas use of standard schemes for scheduling application hints on typical MAID systems is only able to achieve a smaller energy savings of about 25% versus non-MAID storage.

Rapid digitization of content has led to extreme demands on storage systems. The nature of data access such as simulation data dumps, checkpointing, real-time data access queries, data warehousing queries, etc., warrant an online data management

solution. Most online data management solutions make use of hierarchical storage management techniques to accommodate the large volume of digital data. In such solutions, a major portion of the data set is usually hosted by tape-based archival solutions, which offer cheaper storage at the cost of higher access latencies. This loss in performance due to tape-based archive solutions limits the performance of the higher-level applications that make these different types of data accesses. This is particularly true since many queries may require access to older, archived data.

An attractive option for large distributed sites or data centers is to exploit large disk arrays to keep more data in low-latency storage. The decreasing cost and increasing capacity of commodity disks is rapidly changing the economics of online storage and making the use of these large disk arrays more practical. Large disk arrays also enable system scaling, an important property as the growth of online content is predicted to be enormous, both in at-rest storage (terabytes or more) and in delivered data (gigabytes or more per day). The enhanced performance offered by disk-based solutions comes at a price, however. Keeping huge arrays of spinning disks has a hidden cost, i.e., energy. Industry surveys suggest that the cost of powering the nation's data centers is growing at the rate of 25% every year [67]. Among various components of a data center, storage is one of the biggest energy consumers, consuming almost 27% of the total. To make matters worse, increasing performance demands have led to disks with higher power requirements; moreover, storage demands are continuously growing by 60% annually according to an industry report [68]. Given the well-known growth in Total Cost of Ownership (TCO), a solution which can mitigate the high cost of power, yet keep data online, is needed.

Various studies of data access patterns in data centers suggest that on any given day the total amount of data accessed is less than 5% of the total stored [69]. Most energy conservation techniques make use of various optimizations to conserve energy, but this usually comes with a huge performance penalty. Access patterns for certain High Performance Computing (HPC) and enterprise applications have a lot of predictability, and this predictability could be more intelligently used to conserve power. The predictability of data access, or nature of future accesses, could be either learned by the system or be provided to the system by the applications that access the data hosted

on the system. Learning techniques for this purpose have been met with limited success. This difficulty can be attributed to the dynamic nature and different purposes of various applications running on the same system. On the other hand, the idea of applications themselves supplying the hints about their future accesses does not suffer from these limitations. [22] and [70] have explored the idea of using application hints for the purpose of prefetching data ahead of time, thereby reducing the file system I/O latencies.

Our research is centered on the idea of using application hinting on top of MAID systems to build an energy efficient, near-online storage solution. The focus of our research is to identify and address the challenges in building a MAID-based storage solution that can exploit application hints efficiently. The primary challenges that we address in this paper are as follows.

Challenge 1: *Storage Subsystem Architecture to Facilitate Energy Efficiency*

Enterprise and high-end computing environments rely heavily on virtualization of storage resources. Virtualization provides flexibility in resource allocation by decoupling the physical location of the resource from the logical view of the resource presented to the user/server. In other words, many physical disks can be viewed logically as a single large virtual disk. Though it drastically improves the usability of the system, the technique with which virtualization is implemented dictates the performance of the overall system. One of the main challenges in designing a virtualized system is determining the physical placement of the virtual cache/prefetch space. Since the system is virtualized, the cache could be either centrally located or distributed among the disk arrays. Though this choice of centralized or distributed location is not available for the large volume of regular data, both regular and cache data can still be laid out in different ways; i.e., on a collection of one or more single disks, striped across multiple disks within an array, or striped across multiple disk arrays. The choice of cache location and the cache and regular data layouts can have a drastic impact on the performance and energy efficiency of the overall system; hence, identifying the optimal architecture is very critical.

Challenge 2: *Cache Metadata Management*

In high performance computing and enterprise environments, the volume within data sets is huge; hence, any system designed based on caching or prefetching needs to

be able to accommodate an entire working set in cache that is on the order of hundreds of gigabytes to a few terabytes. Since MAID arrays facilitate using disks for caching/prefetching, cache space is not the primary concern. The main problem is the amount of metadata that needs to be maintained in memory on the disk array or metadata server for mapping Logical Block Addresses (LBAs) to Cache Block Addresses. In the worst case, if we were to maintain a one-to-one mapping of LBAs and Cache Block Addresses, the metadata structure will be huge. For instance, consider a storage system containing 500TB of data (part of one or more virtualized LUNs/Address Spaces) with a block size of 32KB. If we decide to maintain a cache of 10%, or 50TB, using the one-to-one mapping would still require close to 15.6 billion mapping entries; that is, one entry for each logical block of the entire 500 TB data set.

A simple lookup table with these entries for the entire virtual address space would be the best option in terms of lookup times, but the size occupied by this structure would be close to 62GB, assuming each address entry takes 4 bytes. As described in Section 3.2, use of other, slightly more sophisticated, techniques would still require a large metadata structure. If one resorts to setting the granularity of data flow in and out of the cache to a fixed-size set of contiguous logical blocks instead of just one, the size of the mapping structure is reduced, but it results in poor utilization of the cache/prefetch space. Hence, we need a more adaptive solution which exploits the nature of data access.

Challenge 3: *Prefetch/Cache Space Management*

Prefetch/cache space is a limited resource. Though MAID uses a large disk-based cache, the use of this cache space is manifold and hence leads to contention. This cache space is used for holding prefetched data blocks from the dormant disks and staged write data blocks heading to the dormant disks from higher-level applications. Since idle periods in servers are used for hint generation, the deadlines of hints could be spread out in time. When a storage subsystem receives I/O hints from multiple servers connected to it, the storage system has to make an informed decision to schedule these hinted requests based on their corresponding deadlines and the state of the cache. Making this informed scheduling choice is a complex task as the state of the system depends on various parameters such as incoming read I/O rate, incoming hint rate, incoming write I/O rate, outgoing write destage rate, the amount of used cache space, etc. Traditional schemes for prefetch scheduling [22] use cost-benefit analysis to try to

maximize the utility of each buffer/cache slot while at the same time ensure fairness of scheduling (i.e., earlier deadlines first). However, in the proposed framework, the objective of the prefetch scheduler is to maximize the time a disk receives to service a hinted request. In doing so, each disk is given a greater chance to work in batch mode by executing requests in a collective manner, which leads to energy conservation. Rest of the chapter is organized as follows - Section 3.1 describes overview of the proposed green store architecture. Section 3.2 details extent-based metadata management. Section 3.3 describes deadline-based prefetch scheduling technique for just in time scheduling of application requests. Section 3.4 details experimental evaluation of proposed system. Section 3.5 concludes with summary and future directions.

3.1 GreenStor Storage System Architecture

Enterprise and high performance computing environments typically make use of a set of disk arrays to satisfy their data storage demands. The storage subsystem in such scenarios is made up of several disk arrays that are connected to clusters of servers by means of a high speed Storage Area Network (SAN). These disk arrays can be connected to the servers in different configurations based on the requirements of the environment. In recent years, virtualization of the storage subsystem has gained importance as it facilitates better management of resources by allowing dynamic addition and removal of storage. Figure 3.1 shows our proposed GreenStor virtualized storage subsystem.

3.1.1 Virtualization Mechanism

Storage virtualization separates logical address space from the physical location of data. In essence, it adds a level of indirection between the file systems on servers or initiators and the storage subsystem. This type of indirection allows the storage subsystem to place, or lay out, the data blocks according to its own knowledge of the environment, and it also provides the flexibility to move data blocks around without having to inform the file system on the servers or initiators. This flexibility is very important for our design considering that locality of data has a strong impact on power conservation and the performance observed by end users or initiators. All requests made by the servers to the storage subsystem are directed to the Storage Virtualization device (SV), since

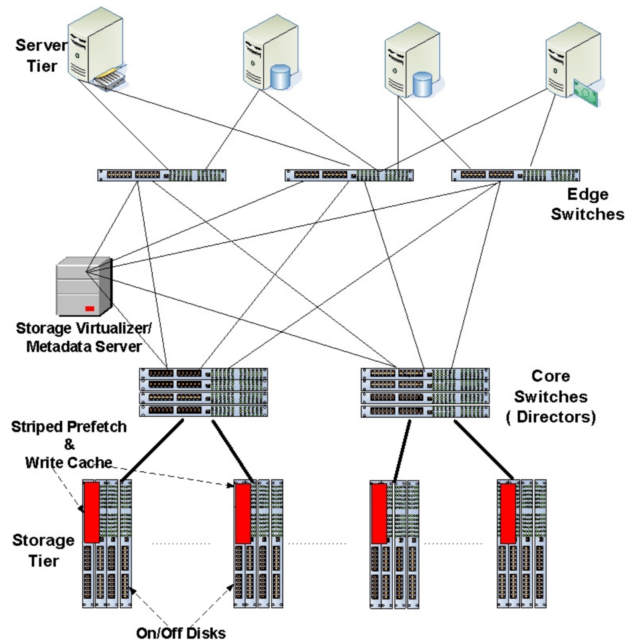


Figure 3.1: GreenStor Storage Subsystem Architecture

this device presents virtual logical volumes, or VLUNS, to the servers. Typical storage virtualization devices can be categorized into two groups; namely, in-band storage virtualization devices and out-of-band storage virtualization devices. With in-band storage virtualization devices, the SV is in the data path. Specifically, all read and write requests are made to the SV, and it services or redirects the requests accordingly. For read requests, the SV fetches the requested data and returns it to the requesting server. For write requests, the SV absorbs the writes and acknowledges back to the servers. The main issue with this approach is that the SV itself could become a bottleneck as it is in the data path. With out-of-band storage virtualization devices, the SV acts like a metadata server; that is, it is only in the control path and not in the data path. When a read request is made to the SV by a server, the SV converts the virtual addresses into physical addresses and returns this information to the server, and the server then uses this information to request the block from the original disk array. In the case of a write request, the same process is followed and the SV remains only in the control path. The main limitation of this approach is that for every read or write request, two

exchanges take place, one between the server and the SV (metadata exchange), and the other between the server and the storage array (data exchange). If network latencies are high, this could severely affect the performance.

The choice of which form of storage virtualization to use is solely driven by performance considerations. This design choice does not have much impact on the energy efficiency of the storage subsystem. We make use of out-of-band virtualization as it is a more scalable solution, and the chances of the virtualization device becoming the bottleneck are reduced as only control information flows through it.

3.1.2 Application Hint Specification

Automatic hint generation is one of the key inputs to our scheme. Hint generation could be performed in several ways. [24] shows that several HPC applications can be modified to generate hints online, specifically, by modifying the binaries of these programs. [23] shows that certain classes of HPC applications can disclose their I/O requirements ahead of time. At the beginning of their execution, these applications could be made to disclose all future I/O accesses along with certain QoS requirements. Some researchers even suggest that application programmers, given the incentive of energy efficiency, could, by themselves, rewrite some of their application code to generate, either online or before execution, hints in advance. Both [24] and [23] show that compilers could be made to embed application hints by speculative execution of code. Our focus is on the efficient use of these hints and not the mechanism of hint generation; hence, we work independent of the mechanism of hint generation. However, we make one key assumption here: Instead of the applications just passing the hints about future access in a sorted manner, we assume that they can be programmed to also pass an approximate time of future access. This assumption is reasonable considering that the applications know their execution rates and can predict their own progress rate better than the storage subsystem, unlike [71] where the disk subsystem infers the application execution rate by monitoring the incoming I/O rate. This assumption is needed in our design because we work from the level of remote consolidated storage that is shared by multiple servers and applications; thus, estimation of the application execution rate becomes very difficult given the fact that most I/O interfaces hide the initiating application information from the target storage devices.

3.1.3 Data/Cache Layout

In a virtualized storage subsystem, the virtual address space can be physically allocated in different ways (usually referred to as LUN binding), namely:

1. Concatenating the address space of the individual Logical Units (LUNs) that constitute the virtual LUN to form a single virtual address space,

2. Striping the virtual address space across the LUNs that constitute the virtual LUN,

3. Using a mathematical mapping function for mapping between the virtual address space and physical address space, or

4. Using random allocation and storing the mapping information for each block in a lookup table.

Options 1&2 are the most popular approaches. Option 1 is the most flexible approach since it easily facilitates addition and deletion of constituent physical LUNs or growing and shrinking of virtual LUNs. Option 2 is very good in terms of avoiding hot spots and provides more parallel bandwidth because the disk accesses are evenly distributed across all constituent LUNs. However, addition or deletion of striped constituent LUNs requires reorganization of a large number of data blocks and hence limits its usefulness. Option 3 is very similar to Option 2 in that Option 2 uses a very simple mapping function (i.e., modulo). Option 3 also suffers from the same drawback of having to reorganize a lot of physical data blocks on expansion or contraction of the virtual LUN. Considering the size of data sets that we consider here, Option 4 is not a feasible option as the size of the lookup table needed to support such a scheme would be prohibitively large.

For our design we have decided to use Option 1, the concatenated address space model, due to the features it offers. It also aids in efficient destaging operation as shown in subsequent sections. The bandwidth is not a major cause of concern here; though we are concatenating address spaces, the constituent LUNs themselves may be internally striped to provide sufficient parallel bandwidth. In our design, each disk array that is a part of this storage subsystem presents one or more LUNs to the SV. The SV

concatenates these LUNs from multiple disk arrays, creates a single large concatenated address space, and presents this Virtual LUN to the servers. A SV may host multiple Virtual LUNs, each of different size.

The prefetch/write cache LUN, highlighted in Figure 3.1, primarily controls the effective read/write bandwidth of the system. Hence, in our scheme, we have decided to use a combination of concatenation and striping across multiple disk arrays to exploit maximum parallel bandwidth. Each disk array that is part of this storage subsystem has a certain set of disks designated as prefetch/write cache disks, which are always on. Each of these disk arrays creates an internally striped address space across its cache disks and presents this LUN to the SV. The SV concatenates these cache LUNs from different disk arrays to create a single large virtual prefetch/write cache LUN. The point to note here is that we force the striping of address space for constituent cache LUNs, but LUNs that constitute data LUNs may or may not striped. It is important that the cache LUNs are striped to improve their performance because they control the effective bandwidth of read and write operations.

3.2 Extent-based Metadata Management

As in any virtualized storage subsystem that uses caching or dynamic movement of logical blocks, the mapping of a given logical block address to its actual location in the cache is a very key operation. The dynamic movement of logical blocks within the system necessitates the maintenance of additional metadata for this remapping operation. The amount of metadata is primarily dictated by the granularity of movement, which in its simplest case is a single logical block. File system block sizes range from 512 bytes to 256KB. Current systems commonly use a block size of about 16KB. However, due to coalescing of writes and read-ahead prefetching implemented by most file systems, I/O requests of about 32KB are very common.

Before going into the details of our mapping method, let us first describe some of the limitations of current techniques. As discussed in the introduction, a lookup table with a one-to-one mapping for the entire virtual address space is prohibitively expensive in terms of its size. A more viable option, in terms of space, is an inverted mapping table, which has an entry for each physical address in the cache. Even for this scheme, a cache

of 50TB with a block size of 32KB would need close to 1.56 billion mapping entries to be stored. Considering the overhead of inverted tables, the space occupied by such a structure would be close to 12.5GB. Also, mapping operations on an inverted table are known to be relatively expensive in terms of computation time. A more commonly used approach for this mapping of logical blocks to their location in cache is to use a hash table. In this scheme, each mapping record would be of the format shown in Figure 3.2.

Logical Block Address(4B)
Cache Block Address(4B)
Pointer to next record(4B)
Parameters for Cache Replacement Algorithms

Figure 3.2: Hash Mapping Record for Extent-based Metadata Management

Collisions in a hash table could be solved using chaining, which adds the additional space overhead, resulting in mapping records of about 12 bytes each. The size of the overall metadata lookup structure would be about $1.56 billion * 12 bytes$, or close to 18.75GB of metadata for a cache size of 50TB. Note here that a record is stored for each block that is currently in cache, just like in the case of inverted tables. Though this amount is not that significant compared to the size of the overall dataset, this metadata structure needs to be kept in main memory that is generally reserved for holding cache/prefetch data blocks, and managing/mapping this record information becomes very cumbersome as the system scales to larger sizes. Also, this new metadata is to be maintained in addition to the metadata already maintained by the file system (using inodes). This new metadata is therefore an added overhead which is used for the sole purpose of caching or redirection. From an initiator or server's point of view, the metadata maintained as part of inodes is inevitable, but this new metadata maintained in the storage subsystem is not easily justifiable.

One solution to this problem of the cache mapping structure being too big is to use a different granularity for movement of blocks from/to cache/prefetch space. [6] uses this approach with a relocation block size on the order of megabytes. Instead of moving one logical block at a time, in this approach, a set of contiguous logical blocks is classified as a relocation group and all data movement is at the level of these larger groups.

This certainly reduces the size of the overall metadata structure, but the utilization of cache/prefetch space drops drastically. The problem here is that normal access patterns usually dictate the choice of the file system logical block size, and this is typically set at the creation of the file system. The values for this file system block size range from 512 bytes to 256KB. Hence, when data is moved into cache at higher granularity, there is high likelihood that only partial hits will be seen, leading to wasted cache space. For instance, if we were to use 1MB as our relocation group size, and if the file system uses a block size of 64KB, in the worst case (i.e., every 16th block needs to be prefetched) we could end up with a cache utilization of 1/16. On the other hand, if the data access pattern is semi-sequential or concentrated around certain regions of the address space, this idea of moving larger chunks of data at a time works really well.

The examples used in the above discussion of the limitations of current techniques for metadata management assume that the average file system block size is 32KB or 64KB. However, this assumption might not always be the case. In systems with smaller block sizes, this problem of metadata management becomes an even bigger challenge as the volume of data grows. The following subsections describe how our extent-based mapping scheme is designed to overcome the limitations of current techniques.

3.2.1 Monitoring Access Patterns

At the heart of our design is the idea of using extents, or contiguous chunks of sequential blocks, to reduce the total metadata overhead. This idea has been used in file systems with a great degree of success. In our design the purpose and use is quite different, however. The main difference is that the selection of extent size at allocation time is not arbitrary, as is the case in file systems. We propose to use a heuristic called *Contiguity Quotient* to guide the selection of extent size. Contiguity Quotient (CQ) tries to quantify the contiguity, or sequentiality of a cache group. Note that a cache group is nothing but a set of contiguous logical blocks of a predefined size. The number of cache groups is equal to the number of blocks in the entire virtual address space divided by the number of blocks in a cache group. We try to estimate the CQ heuristic for each cache group by observing the data access patterns. Specifically, we maintain one access bit per logical block in the cache group, amounting to a 2-byte record per cache group, assuming that each cache group contains 16 logical blocks. In addition,

each record has a Mod Bit to indicate if any modification occurred or not (1 or 0, respectively). Algorithm 1 shows the process for updating the records in the monitoring structure. Monitor Reset Events could be triggered with a predefined periodicity. This

Algorithm 1 Access Monitoring Algorithm

```
Initialize monitoring record: Set Mod Bit of all records to 0
while (1) do
    if (Monitor Reset Event) then
        Set Mod Bit of all cache groups to 0
    end if
    if (Data Access Event) then
        if (Mod Bit of cache group is 1) then
            Set corresponding Access Bit to 1
        end if
        if (Mod Bit of cache group is 0) then
            Set corresponding Access Bit to 1
            Reset Access Bits of all other blocks of this cache group to 0
            Set cache group Mod Bit to 1
        end if
    end if
end while
```

algorithm tries to capture access patterns for a window of time equal to the period of the monitoring events. A count of the number of access bits set in each cache group's record gives the value of the Contiguity Quotient for that group. A higher value of CQ indicates that if a block is accessed in that cache group, there is a very high probability that more blocks in the same cache group will be accessed.

3.2.2 Extent Allocation

If a block request is selected for scheduling, and if this request does not belong to a cache group which already has an allocated extent, the scheduler makes use of the Contiguity Quotient of its corresponding cache group to allocate an appropriate-sized extent for the purpose. Extents are allocated as a best effort service. If an extent big enough for a particular CQ is not available at the time of request, the Mapping Manager, which is a component of the SV, tries to find the best possible alternative for the request. The extent allocation mechanism is very similar to the mechanisms used by most extent-based file systems (ext4, XFS, VxFS, etc.). Free extents are maintained in a B+tree configuration to aid in fast lookup or search. Over time, the cache space could

become fragmented, leading to a large number of small extents. In order to overcome this problem, one would have to defragment the space by moving data around in a way that increases the number of free contiguous blocks, or extents. This process of defragmentation could also involve combining multiple occupied extents with the goal of reducing metadata overhead. Specifically, if a cache group has more than one extent allocated to it (i.e., it is using a secondary record in the Global Mapping Structure), during the process of defragmentation, these occupied extents could be combined to form a smaller number of larger extents, thereby reducing metadata overhead.

3.2.3 Global Mapping

In our design we use hash tables for performing the lookups, but the structure of the record is unique. Figure 3.3 shows the hash table for a single virtual LUN. Every request

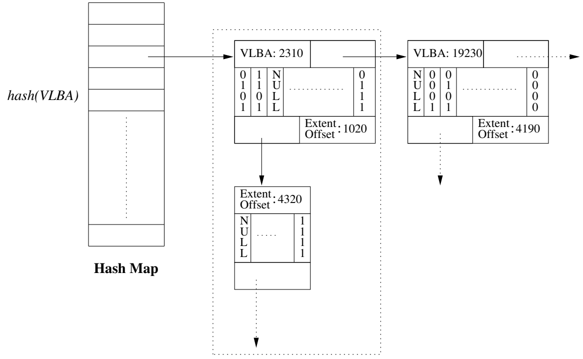


Figure 3.3: Global Mapping Structure for Extent-based Metadata Management

made by any server has a Virtual Logical Unit Number (VLUN) and a Virtual Logical Block Address (VLBA). The VLUN is used to select the corresponding hash table, and then the VLBA is hashed to get the index into this hash table. The hash function we use in our design is,

$$hash_key = \lfloor (VLBA/MAX_CacheGroup_Size) \rfloor \% \beta \tag{3.1}$$

where MAX_CacheGroup_Size is the number of logical blocks per cache group, and β is a variable that controls the length of hash chains or the length of the hash table. Each record contains information about a certain set of contiguous logical blocks. Figure **??**

shows two types of records; i.e., primary records and secondary records. The primary record holds the address of the virtual logical block (VLBA, used for resolving hash collisions), the address in cache of the start of the extent group (extent offset), an array of offsets representing VLBAs stored within the extent (extent offset array), a pointer to a secondary record, and a pointer to the next primary record with same hash value. The number of elements in the extent offset array is determined by the number of blocks that can be accommodated in the extent (which, in turn, is determined by the CQ). In essence, an array element gives the logical address of the block that is present in cache at that extent offset. To better illustrate, consider the first primary record in Figure **??**. The Virtual Logical Block Address is 2310. The extent offset is 1020. The first element (offset of zero) in the array holds the value 0101, or 5. Adding the VLBA of the record to this number would give us the actual VLBA at cache location 1020, so the block of data with VLBA 2315 is stored in cache location 1020. Similarly, the third array element is null and hence the cache location 1022 is free or unoccupied. Considering that we would like to use variable-sized extents, the problem of assigning the correct-sized extent is challenging. Because the extent size is determined by heuristics (i.e., CQ), the result is sometimes imperfect. This leads to cases where two or more extents might need to be allocated for the same block of contiguous logical address space. To handle such cases, we propose using the secondary record structures. The design and function of these structures is very similar to that of primary records with the exception that they do not need to store VLBAs or chain pointers (since collisions would have already been resolved before coming to this stage).

Once a VLBA is checked against the global mapping structure and it is determined if it is cached or not, the following transformations are applied to the VLBA to determine the DA# (Disk Array Number) and the LBA within that disk array:

$$DA\# = \lfloor VLBA/ConstituentLUNSize \rfloor \tag{3.2}$$

$$LBA = VLBA - (DA\# * ConstituentLUNSize) \tag{3.3}$$

where ConstituentLUNSize is the size of the portion of the Virtual LUN that resides in each disk array. Note that the Constituent LUN Size is different for data LUNs and the cache LUNs. This translated information is returned to the requesting server by the metadata component.

3.3 Deadline-based Prefetch Scheduler

Our design heavily relies on the performance of the prefetch or cache space. The size of this cache space is proportional to the size of the entire data set and is accommodated on disk. However, the use or purpose of this cache space is manifold. First, this space is used to store data prefetched from dormant disks using hints from applications. Second, once prefetched into this space from the dormant disks, the data is eventually read from the space by a higher-level server. Third, this space is used for staging write requests destined for dormant disks. Fourth, this data is eventually destaged to the dormant disks. Obviously, these are competing/coordinating factors using the same resource. The optimal use of this resource, or space, is dependent on intelligent arbitration between these competing factors. Here, we model this problem as a classical producer-consumer problem and try to solve the problem with the goal of deep prefetching.

In this model of the system, the prefetch requests and the write requests coming into the system from the servers form the consumer classes. The write destage requests or dirty flushing requests/operations and the read requests are classified as producers because these operations free cache space and generate (produce) a free slot. Here, we make an assumption that once a read operation/request reads a previously prefetched data block from the prefetch space, the corresponding prefetch slot can be freed. This assumption is based on the fact that, in general, almost all initiating servers, or, specifically, the file system components on the servers, implement some form of caching. Hence, if a block is read over the network or SAN, it is usually cached locally by the file system, and any further request for the same block is served using this local copy that is stored in the local cache. Therefore, the assumption that a read request qualifies as a producer of a cache/buffer slot is a reasonable one to make. Further, this assumption can be easily adapted to other situations to implement an approximation of second chance, or even least recently used, behavior by just changing the weight given to this class of producer.

Out of these four types of requests, the prefetch request scheduling is the only one that is under our control. The other three types help in estimating the state of the system only. Specifically, the write requests and the scheduled prefetch requests, which

are consumers of buffer space, along with the read requests for prefetched data and write destage requests, which produce buffer space, help us estimate the load on the system, or the state of the system. By strictly controlling the scheduling of the prefetch requests, one can control the load on the system. Scheduling of these prefetch requests can be performed in multiple ways, namely:

First Come First Served (FCFS): All prefetches that come into the system are executed in a FCFS fashion without any consideration of their corresponding deadlines. This kind of scheduling continues as long as resources, in this case the empty buffer slots, are available. Clearly, this is not an optimal or fair approach because prefetch requests that have more stringent deadlines can be denied service due to their arrival order.

Earliest Deadline First: At any given point in time, all prefetch requests are ordered in increasing order based on their deadlines, and based on the number of empty slots in the system, a corresponding number of prefetch requests are executed in the order of earliest to farthest deadline. This approach is not optimal either. A problem arises because prefetches arrive at random times and there is not temporal ordering between prefetches. This is because multiple servers use the same consolidated storage systems and hence multiple applications generate prefetch hints asynchronously. For instance, consider a case where five buffer slots are available at time $t = i$; using this approach we could end up deciding to schedule the next five requests in the sorted deadline order, say $d_1 = 20, d_2 = 23, d_3 = 35, d_4 = 49, d_5 = 54$. Now, at time $t = i + 1$, if a new prefetch request arrives with $d = 22$, and if no new cache slots are free, we would end up holding back this prefetch request to a later point in time when a buffer slot becomes free. This is clearly not optimal or fair, and in a consolidated storage system, where multiple sources of application hints exist, the problem is quite severe.

Prefetch Horizon: The principle here is that there is no benefit in executing a prefetch request earlier than it is actually required. If the cost or execution time of a regular buffer/cache hit is T_{hit}, and the cost or execution time for servicing a prefetch is T_{disk}, then $p_h = \frac{T_{disk}}{T_{hit}}$ is called the Prefetch Horizon. Specifically, there is no benefit of executing any request that is more than p_h accesses away from the current request. This principle can also be expressed as follows: If a block retrieval time from disk is t_{disk}, and the prefetch deadline of request p is d_p, then there is no benefit of scheduling p until the requesting application progresses to the point $p_h = d_p - t_{disk}$, which is

called the Prefetch Horizon. Prefetching a block after its prefetch horizon does not give any benefit. This kind of delayed scheduling helps achieve a more optimal scheduling and overcomes the shortcomings of the previous two approaches. The objective here is fairness based on deadline. Multiple schedulers like TIP2 [22], TIPTOE [71], and FORESTALL [72] have been designed based on this core idea. These techniques cannot be directly applied to our scenario for a few reasons. First, all the above schemes assume that data accesses can be segregated based on different initiators or application processes. In our scenario, from a storage controller's perspective, all block accesses are the same. It cannot differentiate between applications that are using the storage. Hence, all data accesses need to be seen as a single stream of requests. Any inference of application speed made based on arrival rates of their corresponding I/O requests (as is the case in TIPTOE, FORESTALL, etc.) is not applicable in our setting. Second, in the previously mentioned schemes, application hints are assumed to be an ordered sequence of requests without any explicit deadline. Their deadline is inferred by estimating the application speed. In our scenario, we do not have any explicit knowledge of the speed of the applications that are supplying the hints because our decision making, or scheduling, is happening remotely (in the SAN or storage controller) and away from the servers hosting these applications. Hence, supplying hints without deadlines would not work in our scenario. Consequently, the corresponding scheduling mechanisms are not directly applicable, either. Third, the above schemes consider LRU cache in addition to prefetch cache. In our scheme, since we are at the remote storage system level, we do not have to consider LRU for caching recent accesses by assuming that such caches will be part of the local server cache or file system cache. Fourth, of all these schemes, a couple of them, TIPTOE and Aggressive/LRU, do perform deep prefetching. However, their objective is to minimize stalls caused due to hotspots by making use of idle disks. In our scenario, the objective is different.

Our first objective is to achieve fairness based on deadlines, and the second objective is to give the underlying disk subsystem as much time as possible to execute a prefetch request. The first goal is obvious based on the previous discussion, but the second goal is mainly due to concerns of energy conservation. Since MAID disks are used as underlying permanent storage, in order to maximize energy efficiency, the disks should be OFF as much as possible. If all prefetch requests are issued to a disk well ahead of

time, the disk could potentially make an intelligent grouping of these requests such that it works in batch mode. For instance, if prefetch requests with deadlines, $d_1 = 40, d_2 = 43, d_3 = 60, d_4 = 110$ are queued at a dormant, or OFF, disk, the disk can decide when to turn on based on the nearest deadline, and when it is ON, it can finish servicing the other requests which have less stringent deadlines. By doing so, the disk does not have to stay ON or come back from the OFF state again to service the remaining requests. Conversely, if one uses the prefetch horizon technique, the disk will receive the request with $d = 40$ just before its deadline and will not receive the other requests until their deadlines approach current time. Therefore, this would lead to multiple ON/OFF transitions of the disk, which, in turn, leads to excess power consumption and increases the chances of failure. This goal of energy conservation by batch execution is not easily quantifiable as a benefit model.

With these goals in mind we design a scheduler to achieve optimal scheduling. We try to perform deep prefetching [71]; i.e., prefetching as far into the future as the system permits. In [71] deep prefetching was based on a cost-benefit tradeoff, but here we base it on the resource constraint (i.e., buffer free space availability). We propose a slight modification to the current application hinting paradigms in order to facilitate more optimal performance. Instead of supplying application hints in a sorted order, we propose that the applications also consider their I/O request rate, precalculate the deadlines of prefetch requests explicitly, and supply them to the storage system. By doing so, the storage system can make intelligent scheduling decisions based on deadlines without knowing much about the applications supplying the hints.

3.3.1 Opportunistic Deep Prefetcher

In our approach, when a new prefetch request arrives at the scheduler, it is checked against the Global Mapping Structure to determine if it has already been prefetched or write cached. If not, a check is made to see if any cache extent has already been allocated for the cache group to which the prefetch request belongs. If such an extent is found, and if free space is available within the extent, this request is dispatched to the corresponding disk array along with the address of the free block to be filled. Note that dispatching a request to the disk array or disk does not necessarily mean that the request will be immediately executed by the target. It just means that the target disk is free

to service this request any time before its specified deadline. If the corresponding cache group, and hence the prefetch request block, does not have any mapping in the global structure, the deadline of this request is examined further in order to make a scheduling decision. The scheduling decision for this request is now determined by several factors like the state of the cache and the remaining time between the current time and the request deadline. The objective here is to ensure fairness in scheduling. If this request were to be dispatched, resulting in the use of a buffer slot, and a new request with an earlier deadline arrives soon after, then the earlier scheduling decision should not have any adverse impact on this new request. In essence, if this new request with an earlier deadline cannot be scheduled due to lack of buffer space, then the previous scheduling decision is considered unfair. In order to avoid such scenarios, our approach uses the current state, and estimates of system parameters, to predict the future state. Ideally, we would like to avoid all scheduling decisions that lead to unfair scheduling. This ideal case can only happen if we have perfect knowledge of the system and the system is not dynamic; i.e., properties do not vary with time. In such ideal cases, we could directly use the concept of maximum allocation, like in the Banker's Algorithm, to avoid deadlocks.
Safe & Unsafe States: A state is said to be safe if, and only if, enough buffer slots are available to service all incoming prefetch requests that have a deadline earlier than the current request being serviced. A state is said to be unsafe if the number of buffer slots available is not sufficient to service all incoming prefetch requests that have a deadline earlier than the current request. Here, we assume that the prefetch requests are coming into the system at the rate PRR (Prefetch Request Rate). In the worst case, all the requests that can come between the current time and the deadline of the current request under consideration could have a deadline earlier than the deadline of the current request. In such a scenario, we check to see if we have enough free buffer slots in the system to service $PRR * (d - current_time)$ requests, where d is the deadline of the current request. If the available free space is sufficient to service these requests, then the resultant state is a safe state; hence, the current request can be scheduled for servicing. If servicing of a given request results in an unsafe state, we do not schedule this request. It is held back until the state of the system changes or the deadline becomes closer, thereby increasing the possibility of entering a safe state.

In practice, the deadlines of requests between the current time and the deadline

under consideration may not all lie before the deadline under consideration. Hence, we consider the average-case scenario by using an estimate of the deadline distribution. We obtain a distribution by maintaining a histogram of the deadlines of all incoming prefetch requests. This histogram of the number of requests per deadline can be translated to a probability value per deadline. The deadlines, and hence their probabilities, can fluctuate drastically over time due to variations in application behavior. Using a general averaging or cumulative summation scheme will not reflect the more recent behavior. Hence, for estimation purposes, we use a windowing estimation scheme by successively considering groups of m samples. By using this grouping, we try to ensure that the most recent m samples are considered for estimation of deadline probabilities. Equation 3.4 gives the deadline probability estimation equation,

$$P(d = x) = P_{t-1}(d = x) * \frac{m - n}{m} + P_{current}(d = x) * \frac{n}{m} \tag{3.4}$$

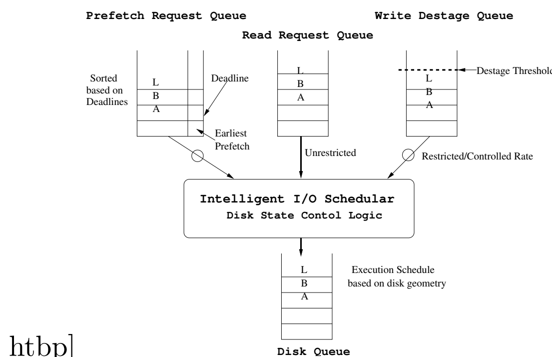
where $t-1$ is the previous estimation window, m is the window size, and n is the number of samples collected in the current window. Now, if a request with deadline d arrives at time t, we estimate the expected state of the system based on the result of Equation 3.6,

$$\begin{aligned} E(U_t) = & \quad (U_t + 1) + d * (PRR * P(deadline < d)) && (3.5) \\ = & \quad (U_t + 1) + d * (PRR * \Sigma_{i=0}^{d} P(deadline = i)) && (3.6) \end{aligned}$$

where U_t is the used cache size at time t. Note that we consider a probability that the deadlines of incoming requests will be less than the current deadline. Using this probabilistic estimate gives us the average-case scenario. Now, if $E(U_t)$ is greater than the cache size, then the system is expected to transition into an unsafe state upon servicing of the current request; hence, this request is not scheduled and is held back for consideration at a future time instant. On the other hand, if $E(U_t)$ is less than the cache size, the system is expected to remain in a safe state and hence the current request is scheduled immediately (i.e., dispatched to disk for servicing).

3.3.2 Delayed/Lazy Execution

From an energy conservation point of view, a disk should be in its Standby or OFF state as much as possible, and the number of transitions between ON and OFF or Standby



htbp]

Figure 3.4: I/O Queueing Model for Delayed/Lazy Execution

states should be minimized. If the disks were to service each and every prefetch request immediately, without considering the deadline, or service write destage requests when they arrive, our objectives of energy conservation would be violated. However, disks need to immediately service real-time read requests or requests which have not been hinted or prefetched. For the two non-immediate classes of requests, we try to develop a delayed, or lazy, scheduling mechanism based on intelligent queuing.

Disk Queuing: Standard disk queuing schemes use certain physical characteristics of the disks to arrive at an optimal access schedule. For our design the focus is not these disk scheduling schemes. We focus on adding more intelligence to the storage controllers for the purpose of energy management. Specifically, the storage controller maintains a set of logical queues for each disk drive. The properties, or dynamics, of these queues influence the power management decisions. Figure 3.4 shows the queuing model for a single, non-cache, MAID disk in our system.

The Write Destage Queue (WDQ) holds all destage requests sent by the Mapping Manager to the disk. These requests do not have any specific deadline or scheduling order. The Prefetch Request Queue (PRQ) holds all prefetch requests dispatched by the mapping manager to the disk. This queue is sorted based on the deadline of prefetch requests. The Read Request Queue (RRQ) holds all real-time read requests for blocks on the disk. These requests do not have any specific order, but are to be considered the highest priority.

The final goal of this queuing system is to ensure that energy consumption of the disk drives is minimized without sacrificing real-time performance and that the deadlines of

prefetch requests are met. Specifically, we develop a power management scheme which works according to the status of these queues. We develop a set of rules which govern the power management, or transitioning, of disks. A disk can make a transition to its Standby state (or OFF state) from its Idle (i.e., ON but not Busy) state if, and only if, all of the following conditions are met:

- RRQ is empty,
- Deadline of the prefetch request at the front of the PRQ is well beyond the disk transition time (ON to OFF plus OFF to ON again),
- Depth of WDQ is below a specified threshold, and
- Disk Idle Time, or time since the last real-time read access, is above a specified threshold.

A transition from the Standby or OFF state to an Active (Busy) or Idle state can be made if, and only if, one or more of the following conditions are met:

- A new Read Request arrives (RRQ is no longer empty),
- Deadline of the prefetch request at the front of the PRQ is close to the disk transition time,
- Depth of WDQ is above a specified threshold, or
- Time since the last transition into Standby or OFF state is greater than a specified threshold.

The last condition for a transition from a Standby or OFF state to an ON state is purely from the standpoint of minimizing the risk of disk failure. Studies have shown that if a disk drive is in an OFF or Standby state for too long, moisture from humidity can set in and increase the probability of disk failure on the next transition. [7] overcomes this issue by using a similar technique called Disk Jogging.

3.4 Preliminary Results

3.4.1 Comparison of Scheduling Schemes

For the purpose of comparison and quantification, we compare our opportunistic scheduling scheme with a pseudo prefetch horizon scheme. We call it a pseudo prefetch horizon scheme because the simulation is conceptually similar, but not identical, to the original scheme. We do not consider a First Come First Served (FCFS) scheme because, as expected given the delayed/lazy disk scheduler, the power consumption of this scheme was found to be the same as that of our scheduling scheme. FCFS fails when it comes to fairness, however, as it does not arbitrate based on deadlines while scheduling a request. We do not consider traditional MAID systems (like Copan MAID) for the comparison as prefetching in response to application hints is not a part of these original MAID designs and hence would not be a fair comparison. Hereafter we consider only the prefetch horizon scheme for comparison. For all of the following experiments, we keep the simulation workload constant as performance numbers are gathered. All the results reported here are averaged over five runs. The standard deviations are very low; hence, the graphs do not include this measure.

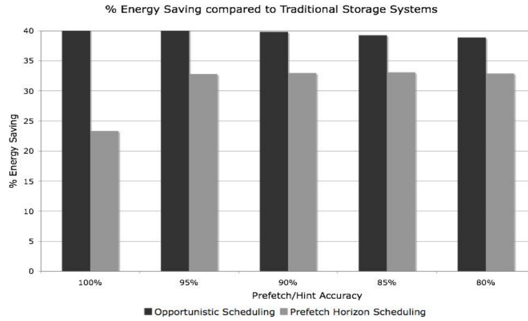


Figure 3.5: Impact of Scheduling Schemes on Percentage Energy Savings

Figure 3.5 shows the plot of percent energy savings for the two schedulers with varying prefetch/hint accuracy. In this experiment, we measure the energy consumption of a traditional storage subsystem when executing the fixed workload and then perform the same measurements with the two scheduling schemes. As expected, for our scheme, the

case with 100% accurate predictions yields the maximum energy savings. Our opportunistic scheduling scheme coupled with delayed scheduling yields close to 40% energy savings. The prefetch horizon scheme results in an energy savings of about 23%. As the prefetch/hint accuracy reduces, the energy savings also drop slightly for our scheduling scheme. This is exactly as expected, because with a reduction in prefetch/hint accuracy, more read requests need to be serviced by the dormant disks instead of the cache disks. This reduces the time these dormant disks spend in their Standby/Sleep state and thereby increases power consumption. However, in the case of the prefetch horizon scheduling scheme, the energy savings actually increase as the prefetch/hint accuracy reduces. This is slightly counterintuitive. Upon further investigation, we find that the main reason for this increasing energy savings is that the number of disk restarts in this case decreases with the reduction in prefetch/hint accuracy.

Deeper examination of the disk states and time spent in each state reveals several interesting facts. We classify the time spent by the disks into idle state, busy state, and sleep/standby state. The time spent transitioning from sleep/standby to idle/busy and vice versa is counted as part of the time spent in sleep/standby state. We find that our scheduling scheme minimizes the time spent in the idle state (i.e., the state where disks are spinning but inactive) at the cost of making more disk transitions, or restarts, to and from Sleep/Standby. This is mainly attributed to the opportunistic behavior of the scheduler, which gives the disks more time to service a request. Another interesting result is the fact that our scheduling scheme minimizes the time spent in the Idle state to less than 20%. Ideally, one would expect to see much higher energy savings in Figure 3.5 than what is shown. The reason for this behavior is the large number of disk transitions that occur with our approach, as shown in Figure 3.6. Disk restarts have a very high energy penalty. Typically, the Idle state power consumption is about 6-8 watts, while the peak spinup power consumption is about 15-20 watts. In addition, the cost of keeping the set of cache disks always powered on further reduces the energy savings. In the prefetch horizon case, the percent of time spent in the Idle state is still very significant, about 50%. The main reason for this is that the prefetch horizon scheme tends to dispatch a constant stream of requests to the disk and does not allow the disk much flexibility in serving these requests. One interesting behavior to note is that, in prefetch horizon, with a decrease in prefetch/hint accuracy, the percent

of time spent in the Idle state increases. The main reason for this behavior is that the disk gets fewer chances to transition into its Sleep/Standby state because both unhinted and hinted prefetch requests arrive at the disk at a steady rate that needs to be serviced immediately.

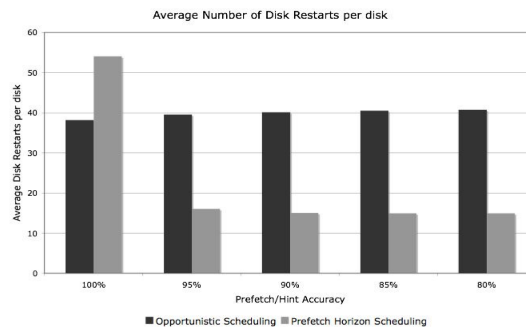


Figure 3.6: Impact of Scheduling Schemes on Average # of Disk Restarts

Figure 3.6 shows the average number of restarts, or transitions, per disk for each of the scheduling schemes. With 100% accurate predictions, the number of disk restarts is about 50 for the prefetch horizon scheme. This is slightly higher than that of our scheduling scheme, where the average number of disk restarts is about 40. This is to be expected as our scheme tries to dispatch requests well in advance in order to maximize the time spent in the sleep/standby state. When moving away from perfect prefetch/hint accuracy, the number of disk restarts in the prefetch horizon case drops drastically, whereas it is relatively constant in the case of our scheduling scheme. This, in turn, leads to less time spent in the Standby/Sleep state when using the prefetch horizon scheme. Because the energy saved by avoiding disk restarts is considerably larger than the energy lost in this reduction of sleep time, we see this interesting behavior where the prefetch horizon scheme's energy savings increases as the hints become less accurate.

Figure 3.7 shows the read performance of the system with varying prefetch/hint accuracy. For the purpose of this study, our main interest with respect to performance is determining if a request was served from a cache disk, a dormant disk that was on, or a dormant disk that was off. Hence, we classify the read responses into two categories; namely, requests with response time in the milliseconds range and requests with response

time in the seconds range. With 100% accurate predictions, all the read responses seem to be completed in the millisecond range for both approaches. As prefetch/hint accuracy decreases, the percentage of requests completed in the millisecond range is smaller in the case of our scheduling scheme compared to the prefetch horizon method. Again, as shown in Figure 3.6, the disks spend less time in a Sleep/Standby state with the prefetch horizon scheme as a consequence of a smaller number of restarts. Hence, when an unhinted request arrives, the probability that the corresponding disk is not in the Sleep/Standby state is higher for the prefetch horizon scheme, and indirectly results in better read performance than our approach.

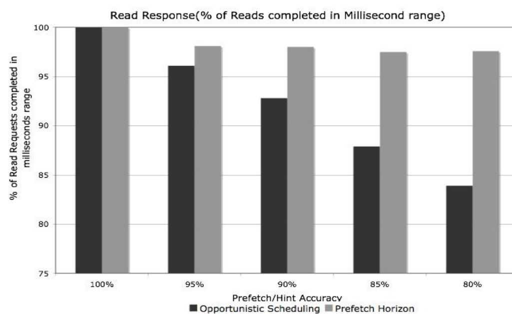


Figure 3.7: Impact of Scheduling Schemes on Read Response Times

3.4.2 Comparison of Metadata Management

Figure 3.8 shows a comparison of different cache/prefetch block allocation schemes. We compare our extent-based allocation scheme, which uses CQ to dynamically determine the size of the extents, with a traditional one-to-one mapping scheme and with a scheme that uses a larger fixed granularity; i.e., it stores fixed-size groups of 4, 8, or 16 blocks (labeled FG(4), etc., in the figure). For this simulation, we generate a synthetic workload consisting of logical block requests with an average cluster size of eight contiguous logical blocks. Out of this synthetic workload we randomly pick a logical block cluster and try to allocate appropriate space in cache using different allocation schemes. This random sampling process is continued until the cache is full, at which point we calculate the performance results. Since our approach makes use of a heuristic estimate of contiguity

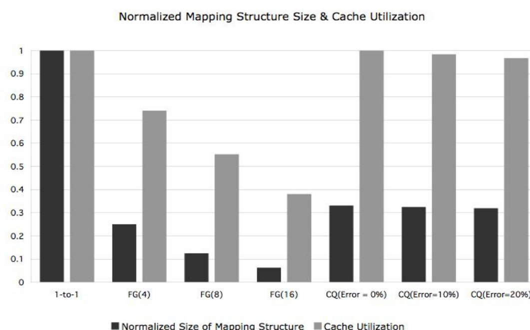


Figure 3.8: Impact of cache/prefetch block allocation schemes on Metadata Efficiency

within any cache group, we also inject different percentages of error into our contiguity estimates to simulate real system behavior where access patterns can change over time. The size of the mapping structure is calculated based on the number of extents, or groups, that fit in the cache. The one-to-one method uses an inverted mapping, so its mapping structure size is directly proportional to the number of blocks that fit in the cache. The FG(4) method creates one fourth as many mapping entries, so its normalized size is 0.25, as is seen in Figure 3.8. However, because unneeded blocks are sometimes brought into the cache when a fixed-size group is used, the cache utilization is worse. This pattern continues for FG(8) and FG(16). Our CQ-based extent mapping method stores more information in the Global Mapping Structure, so each cache mapping entry is assumed to be about twice as large as the one-to-one mapping entry (16 bytes vs. 8 bytes). The figure shows that our mapping structure uses a little more space than a fixed-size group of four blocks. However, our cache utilization, which is above 90% even with slightly erroneous CQ estimates, is higher than any of the fixed-size methods. This shows that we are able to substantially reduce the size of the mapping structure without losing very much in terms of cache utilization. For a more detailed version of this work the reader should refer [73, 74].

3.5 Conclusion & Future Work

In this work we have clearly shown that energy savings can be substantial when application hinting is used on top of MAID storage systems. On average, the results show that the energy savings achieved by using even the existing scheduling techniques is greater than 25%. Scheduling technique used drastically impacts the amount of energy consumed by the system. On average, our GreenStor storage system using its opportunistic deep prefetcher provides energy savings of an extra 10% or more compared to other scheduling techniques. When prediction accuracy is very high, a system with our scheduling technique consumes about 40% less power than traditional storage systems without any prefetching.

In this work we do not consider any constraints on disk transitions. It has recently come to our attention that, in MAID systems, disks are packed every closely to minimize the impact of humidity on disk lifetime. Specifically, if disks are not turned on for a long period, the probability of failure on the next spinup increases due to the collection of moisture from humidity. Hence, MAID manufactures pack disks closely to minimize the effects of humidity. In our proposed design, this close packing of disks could lead to overheating in certain regions of the disk array if not properly arbitrated. Approaches by which intelligent arbitration of disk spinups could be used to minimize the problem of overheating is an interesting avenue for future work.

Chapter 4

Backup Characterization

Data protection is a critical aspect of every modern enterprise. Exponential growth of information, shrinking backup window due to 24X7 nature of global business structure, scaling and commoditization of primary systems all have put increased stress on data protection systems from two perspectives: i) scalability ii) dynamic adaptability. Consequently leading to degradation in operational efficiency of data management. In order the better understand the reasons for inefficiencies, we analyze configuration and monitoring data from data protection infrastructure across several production datacenters over a two month window and characterize configuration and workload variety. Using this data driven approach, we identify key challenges and inefficiencies in current backup ecosystems and consequent limitations in improving operational efficiency and scaling such systems.

Exponential data growth, shrinking backup windows pose significant challenges for traditional backup/data protection systems. Traditional data protection solutions follow a hardwired model of clients, servers and backend devices where clients are installed with a backup agent that interacts with backup servers. Backup servers process and store the data on the backend devices along with metadata. Backup policies are set apriori with respect to policies and schedules without any adaptability to the backup workload variations. Throughout this chapter, the terms backup and data protection are used interchangeably.

Such static paradigms lead to significant inefficiencies with improper resource utilization, performance hotspots for backup server(s) and eventually poor business continuity

Service Level Agreements (SLAs) for modern enterprise. With wider adoption of devops model, more and more of system management actions are now exposed to application developers. As a result, rate of changes of systems and their configurations is increasing rapidly and so is the need for more fine grained (time) data protection. Cloud based data protection as a service have been widely adopted [75].

Backup servers multiplex its compute, network and storage resources across two broad categories: i)client interactions ii)internal data and metadata organization. So, cycles for performing regular internal background tasks such as indexing, deduplication, etc.. are getting impacted on multiple fronts since not only is the amount of data to backup every day growing, but also the number and types of background functions being introduced on backup servers is also growing severely impacting maintenance windows. Similarly, static backup policy setting does not align with the dynamic workload pattern in a virtualized environment.

In order to keep pace with rest of the rapidly evolving ecosystem - applications, workloads, and servers, there is an inherent need to re-architect the old traditional model of backup to overcome the inefficiencies. A holistic change can be brought into the environment with a distributed architecture in mind that spreads the backup workload between clients and servers, among multiple backup servers, and adapts the backup policies in an autonomic fashion. Emergence of virtualization technologies and cloud based data protection solutions also perfectly aligns with such an architecture.

Multiple studies have focussed on characterizing backups workloads [35] [43]. [37] provides a characterization of storage configuration variety and volume on virtual machines. However, very little literature exists on variety of policy and storage configurations across data protection systems.

Employing a data driven approach, we characterize configuration and workload variety of backup servers and identify key challenges in current backup ecosystems and consequent limitations in scaling such systems. Rest of the chapter is organized as follows - section 4.1 discusses data collection methodology, section 4.2 details configuration variety, section 4.3 highlights workload variety and sections 4.4, 5.5 discuss lessons learnt and challenges/inefficiencies identified and conclusions.

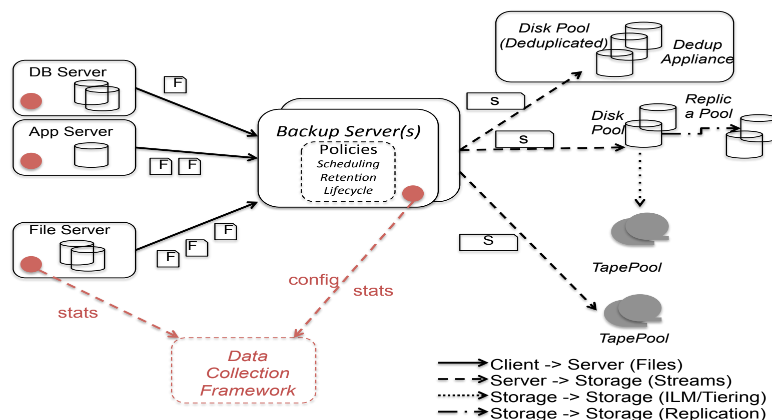


Figure 4.1: Data Collection from Traditional Enterprise backups

4.1 Data Collection

For IT service providers, data protection services are one of the key services offered. Data protection as a service is also offered in a siloed fashion oblivious to server and storage management. For Monitoring, compliance verification and enforcement, backup servers managed by data protection services send backup meta data to a central warehouse as shown in Figure 4.1. Monitoring information is collected at varying schedules ranging from hourly to twice a day based on the backup window spread of the managed environment. We instrumented client systems and extracted about 60 days worth of this monitoring metadata from central warehouse of an IT service provider for our analysis from 2013. This monitoring data contains metadata records of each backup performed in these environments. Metadata captures a wide spectrum of information. From our analysis view point, we extracted the metadata that includes the client configuration details, the backup server configuration details, size of backup, # of files, the type of backup, the start time, the time of completion and the storage hierarchy behind backup server. For our data characterization, backup monitoring data is not sufficient by itself. For more detailed configuration level information(policies, storage and hardware configuration), we instrumented our data collection framework for a subset of these backup servers (about 2000). Selected backup servers are in different data centers dispersed around the globe, serving many customers from a wide variety of sectors/industries. This subset of backup servers account for about 2 million backup jobs on a daily basis.

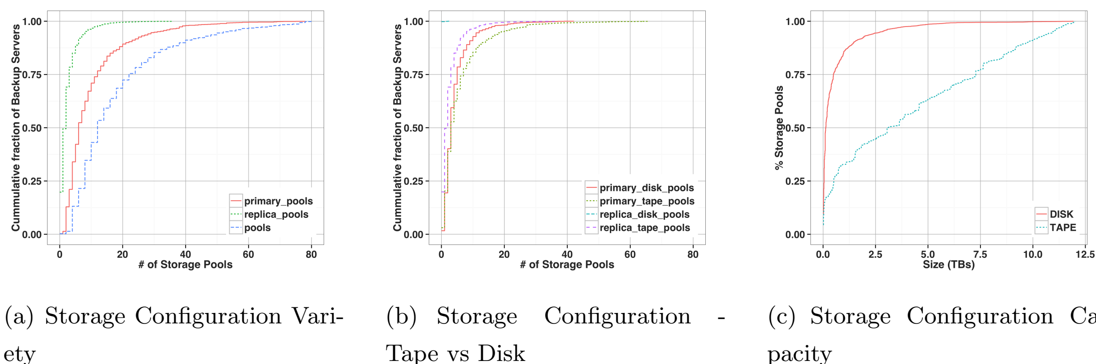
This configuration data includes client to policy associations, policy to storage associations. Since this configuration data varies less frequently and involves more intrusive querying , we limit this collection to weekly runs.

4.2 Configuration Variety

Backup configurations vary widely across different environments. Some configuration variations such as storage configurations and client associations are driven by scale considerations where as variations in schedules and backup windows are driven by admin behavior. In this section we characterize these different types of configuration variety across all backup systems in our dataset.

4.2.1 Storage Variety

Storage configurations of backup servers are dictated by various factors such as - cost, scale - backup/restore performance, and multi-tenancy. Figures 4.2(a) 4.2(b) 4.2(c) shows storage configuration variation across all backup servers. Figure 4.2(a) shows a cumulative distribution of number of storage pools across backup servers categorized based on different types of pools. About 45% of backup servers have less than 10 storage pools in total. Conversely, about 55% of backup servers profiled have more than 10 storage pools for a client to choose from for backup or archive purposes. This high number of pools per backup servers illustrates the complexity of multi-tenant or cost based policies in place in different environments. About 30% of backup servers have greater than 10 primary pools, where as only about 10% of backup servers have similar number of replica pools. Replication for redundancy at pool level seems to be only in use in a limited set of environments (1/3rd). From an administrators perspective, segmenting resources into distinct pools achieves goals of isolation and multi-tenancy, but from a system performance optimization perspective, such fine grained segmentation can lead to suboptimal resource utilization or load imbalances especially if given static policies and if administrations are carefully considering utilization across these pools which determining placement of new backup loads. Figure 4.2(b) shows a cumulative distribution breakdown based on type of storage device. About 95% of backup servers have less than 10 disk based primary pools, where as a less percentage 85% have less than



(a) Storage Configuration Variety

(b) Storage Configuration - Tape vs Disk

(c) Storage Configuration Capacity

Figure 4.2: Variation in Storage Configuration

10 Tape based pools. One would assume that disk based pools would be more prominent as primary pools given the performance characteristics, but based on the data, Tape based pools seems more in number. Possible explanation could be that tape cardinalities are typically higher than disk based storage volumes or cost dynamics still make tape a more popular choice than disk in environments analyzed. Almost all secondary pools seen were tape based pools which could be driven by cost considerations.

Capacities of different types of pools varies significantly. Figure 4.2(c) shows cumulative distribution of comparison of capacities of disk and tape base pools. Disk and tape pools differ significantly in terms of their configured capacities. 75th percentile of disk pools size is about 1TB , where as tape pools tend to be much larger and corresponding size is about 7.5 TB. This can be mainly attributed to the fact that tape pools typically share a common pool of scratch tapes. Our data does not have enough information to discount this common scratch capacity, however, overall tape capacity at a backup server level is significantly more than corresponding disk capacity. We believe this behavior is dictated by cost dynamics and potability of tape media (offsite vaulting).

Storage Tiering

Business value of backups vary across their lifetimes. Most recent backups tend to be considered more business critical from a recovery from logical failure perspective where as from a compliance perspective, recency does not necessarily have a familiar impact. As described in the background section, backup servers implement various lifecycle

MEDIA TYPE	CHAIN_LENGTH 0	1	2	3	4
DISK	17.22	79.5	3.07	0.11	0.01
TAPE	95.21	4.51	0.28	NA	NA
OVERALL	69.51	2.93	0.01	0.00001	0.000001

Table 4.1: Storage ILM Chain Length

management strategies with tiering being the most prominent one. Tiering typically involves chaining storage devices in certain order with specific policies governing movement between these devices. Most commonly, threshold based policies are used, i.e., move from pool A to pool B when pool A utilization becomes greater than 90%. We analyzed length of such chains and table 4.1 shows a breakdown of chain lengths based on media type of Pool. Chain length of 0 indicates no chaining, .i.e., backups destined for this pool stay inlace until retention thresholds are met, after which they are deleted. For disk based pools, majority of the pools have a chain length of 1 (close to 95% of pools). On the other hand, majority of tape based pools have a chain length of 0. This illustrates the typical tiering behavior in backup servers, where disk pools are used a staging area or as destination for most recent backups and are then destaged to Tape based pools based on capacity thresholds or retention requirements. In about 3-4% of disk pools, chain length is as high as 2. In a very small subset of disk pools ¡ 0.01%, chain length is about 4 - backups destined for this pools could potentially move across 5 different pools over its lifetime.

4.2.2 Association Variety

Backup client to backup server associations are typically statically defined as provisioning time. Decision of which client backs up to which server is done by domain experts based on several parameters. At a minimum, administrators consider the number of configured clients per backup server with the goal of balancing load across backup servers.

Based on capability of backup servers and number of configured client associations, typically backup servers are categorized into X-SMALL, SMALL, MEDIUM, LARGE,

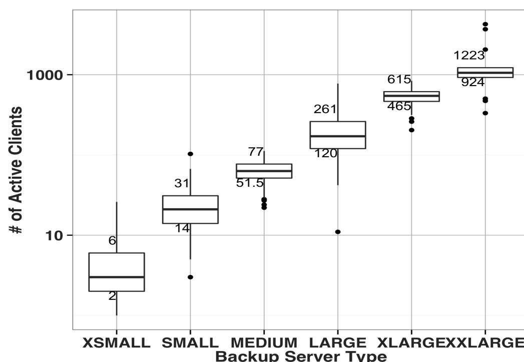


Figure 4.3: Variation in Client-Backup Server Associations

X-LARGE and XX-LARGE with number of configured client associations - <10, [10-50],[50-100],[100-500],[500-1000], >1000 respectively. We categorized backup servers in our data set based on similar industry standard classifications and Figure 4.3 shows a box-plot of average active clients across these categories of backup servers on a daily basis.

Figure 4.3 shows variation in number of active clients for each of these categories. Since distribution is long tailed, the first and third quartiles are highlighted. Considering the third quartile, roughly 60-70% of the configured clients are active on a daily basis. Further, this activity percentage varies widely across backup servers and also within backup server across different days. From a performance optimization standpoint, ideally static associations should be reduced or eliminated. Associations should be dynamically decided based on current load. Given technology & complexity limitations (federating backup catalogs), administrators should use more fine grained activity profiles than solely relying on number of configured clients while defining static associations.

4.2.3 Backup Window Variety

Scheduling of backups can be categorized into predefined window based and adhoc user driven scheduling. In this section we analyze predefined scheduling across the backup dataset. Figures 4.4, 4.2.3 show backup window variations across all backup servers. Figure 4.4 shows start times of backup windows. Based on category of backup server

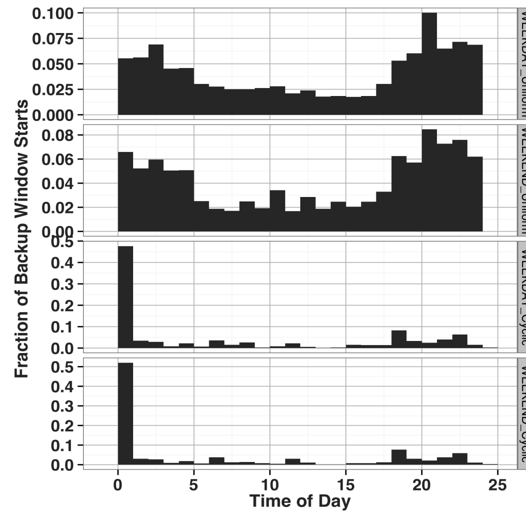


Figure 4.4: Variation in Backup Window: Start Times

and weekend (Fri/Sat/Sun) or weekdays, start times are grouped into four categories - weekday_uniform, weekend_uniform and weekday_cyclic and weekend_cyclic.

Across all groups the start times are more skewed towards later hours of the day or are concentrated towards early hours of the day. Peak activity is concentrated between 20:00 to 05:00 for both types of backup systems. For cyclic systems, about 50% of backup windows start around the same time around midnight. For uniform systems, a non-negligible portion of clients have their backup windows start during the day. Further, given more full backups happen during weekends (for cyclic systems), one would expect start times to be different on weekends. However, the data shows that window start times are consistent across days. This could be a result of administrators selecting a single static strategy per client for simplicity reasons as determining best start time is complex and involves coordination between application owners and backup administrators.

Figure 4.2.3 shows configured duration of backup windows. For uniform systems, roughly about 50% of backup windows are shorter than an hour. Since these systems mainly perform incremental backups, shorter duration expectation is understandable. About 25% are between 5 hours to a day. For these windows, even if start times start during off-peak times, configured maximum duration can potentially carry it through

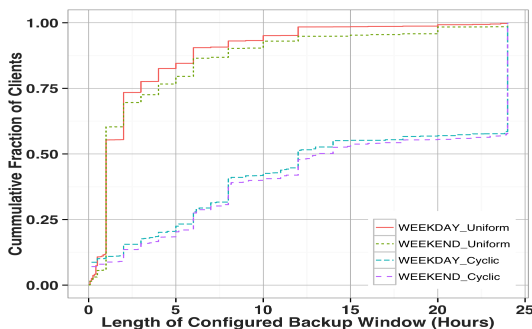


Figure 4.5: Variation in Backup Window: Duration

the day. For cyclic systems, we find that the windows are much larger. Close to 40% windows for cyclic systems have a duration of 24 hours. On further investigation we found that best practices on these cycle systems recommend configuring a large window so as to all flexibility to the servers for scheduling (given that both incremental and full backups are employed on these systems). On both types of systems, we do find limited difference in duration between weekdays and weekends.

4.3 Workload Variety

Backup workload is a function of cumulative churn of systems being backed up and type of policies in place. In this section, we characterize variation in backup workloads across days using 60 day historical metadata, specifically analyzing variation in daily backup server ingest, variation based on backup type and variation within backup clients.

4.3.1 Backup Server Ingest Variation

Backup servers ingest significant amount of data on a day to day basis equivalent to total churn in the environment that they are backing up. Daily ingest amount heavily influences design of staging spaces and capacity planning of primary pools. Figure 4.6 shows a box plot of average daily ingest grouped by day of the week across set of all backup servers in the dataset. Median values are highlighted for each day. Given variation in behavior of uniform and cyclic data protection systems, we categorize the statistics separately. For uniform systems, variation between different days of week

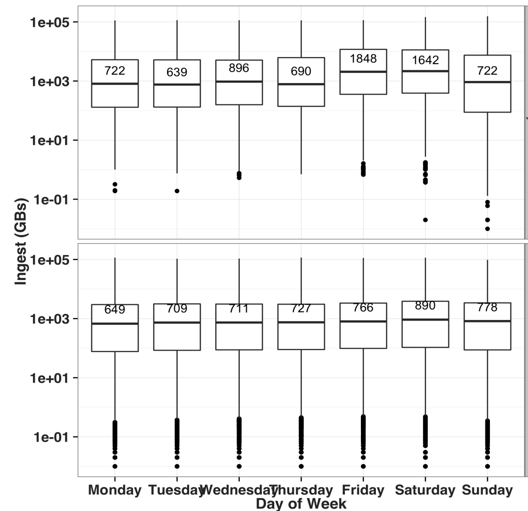


Figure 4.6: Breakdown of Backup Server Daily Ingest

are minimal. Variation between weekdays and weekends is less than 10%. On the other hand, for cyclic systems, the difference between weekdays and weekends is almost 100%. This large difference confirms the scheduling behavior of incremental backups on weekdays followed by full backup on weekends.

4.3.2 Backup Type Variation

Each type of backup has unique signature in terns of size of backups and number of files in each backup. Table 4.2 shows a breakdown of backup characteristics broken down based on backup type. Since these statistics are typically long tailed, the table provides first & third quartile along with mean and medians for both Size and File Count. Comparing Incremental and full backups, median size of full backup is almost 11 times that of incremental backup. This indicates that daily churn captured by incremental backups is roughly about 9 -10%. Comparing differential and full backups, average size of full backup is about 4 times that of differential backups. Manual backups & archival transfer significantly less amount of data than other backup types. Partial or log only backups are significantly smaller as only log files are backed up and pruned regularly.

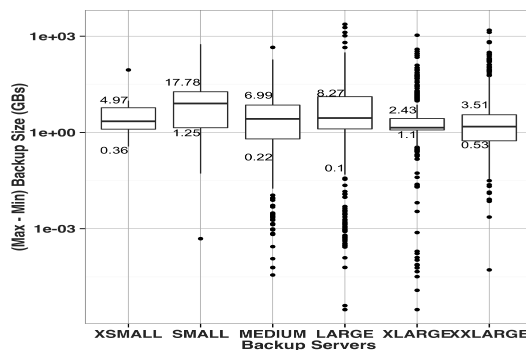
			Size (GBs)					File Count			
Backup Type	Product Type	Description	Q1	Q3	MEDIAN	MAX	MEAN	Q1	Q3	MEDIAN	MEAN
Full	Uniform	Once a week backups	0.16	29.97	4.21	21023.51	63.76	1	9	4	8
Full	Cyclic	Once a week combined with weekday incrementals	1.01	33.52	8.82	17638.70	74.80	80	91253	15616	169602
Incremental	Uniform	Weekday and weekend incrementals	0.14	1.99	0.69	23869.44	7.54	98	801	300	4787
Incremental	Cyclic	Weekday incrementals	0.07	2.41	0.76	8824.88	8.36	43	1517	344	6610
Differential	Uniform	Weekday differentials	0.00	1.93	0.06	4678.81	9.90	1	12	5	11
Differential	Cyclic	Weekday differentials	0.50	5.72	1.62	6340.28	17.55	215	3484	673	20171
Manual	Uniform	User defined backups	0.00	0.20	0.02	5996.61	5.27	1	5	1	393
Manual	Cyclic	User defined backups	0.02	1.23	0.28	16241.64	5.87	1	5	3	1394
Archival	Uniform	HSM archivals	0.00	0.12	0.03	39571.49	3.36	1	4	1	244
Archival	Cyclic	HSM archivals	0.01	0.98	0.15	634.97	3.63	1	9	2	43
Partial	Uniform	Log only backups	0.00	0.01	0.00	2679.19	0.24	1	7	3	7

Table 4.2: Backup Workload Statistics

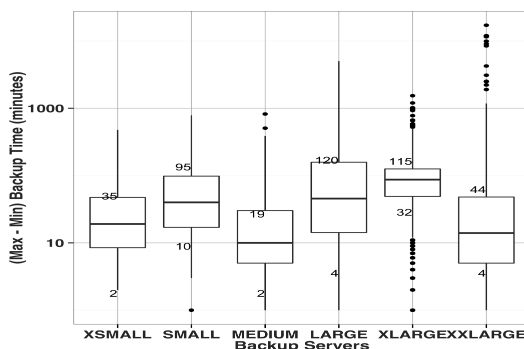
4.3.3 Client Workload Variation

For any given client, the daily churn and correspondingly the duration of backup can vary widely. Figure 4.7(a) shows a box plot of variation in backup data size on a daily basis over a 60 day analysis window grouped by their corresponding servers. The x-axis shows selected set of 6 representative backup servers one per each category. Y axis shows the difference in backup size (MAX-MIN) or churn range for each client aggregated over the entire backup server. Since distributions are long tailed, both the upper and lower quartile values are highlighted for each server. For XXLarge,XLarge servers the inter quartile range is quite minimal - 2.9 and 1.3 respectively. For Small and medium servers highest variability - 16.5 and 6.77 respectively can be seen.

Given these variations in churn, durations of backups should also exhibit similar behavior. Figure 4.7(b) shows a box plot of variation in backup completion time on a daily basis over a 60 day analysis window grouped by their corresponding servers. Y axis shows the difference in backup completion time (MAX-MIN) for each client aggregated over the entire backup server. Highest inter quartile range is for Large and Small backup servers - 116 and 85 respectively. XXLarge and Medium servers show a



(a) Size Variation



(b) Duration Variation

Figure 4.7: Variability across Backup Client Workloads

relatively modest range - 40 and 17 respectively. This duration variation and the size variation do correlate in some instances. For instance for Small servers, both the size churn and duration variation are very high. This could be an indication of overloading or poor load balancing of these servers. However, this correlation is not always applicable is is heavily dependent on backup server utilization state and current load.

4.4 Discussion

Scalability and cost considerations of next generation computing environments necessitate a rethink of data protection systems as well. From analysis of several production environments, we derive the following important insights that impact future backup

system design -

1. Collective churn or volume of data backed up per day across all clients backing up to a given server is far outpacing growth in backup server resources from a bandwidth perspective.

2. Backup window violations will become the norm if backup server resource utilization is not balanced or backup server resources are not over provisioned sufficiently.

Space saving technologies such as compression and deduplication do help address some of these issues, but a broader rethink is required to address these alarming trends. Possible additional alternatives include -

1. Exploit local capacity/bandwidth via hybrid backup architectures that combine local backup,peer-2-peer and centralized backups. [37] found average occupancy/capacity utilization on servers to be about 45%. Local disk bandwidths often exceed average backup system bandwidths. One probable approach is a solution that seamlessly makes use of local disk capacity/bandwidth for temporary staging of backups and then de-stages these backups to central backup servers based on resource availability, thereby reduces the need to peak provision/scale backup servers and backup window issues. However, this solution will require careful modeling of Recovery Time (RPO) and Recovery Point Objectives (RTO).

2. Enabling load based dynamic scheduling of backups. Instead of relying on static scheduling of backups, making backup scheduling decisions dynamic can help significantly reduce contentions between backup jobs and help stretch scalability of existing backup systems.

3. Enabling dynamic associations/load balancing by federating backup catalogs across clusters of backup servers. Cloud based data protection as a service aligns well with this model. Some commercial offering already attempt to exploit benefits of clustering. But the lack of fine grained optimizations/load balancing leaves lot to be desired.

4. Enhancing Backup server architectures with SSDs to improve backup server Ingestion rates. Since most IO in backups systems first goes to scratch/buffer areas,

one possible alternative is to enhances these areas with SSDs that deliver order of magnitude better bandwidth than traditional disks.

In summary, our exploration shows significant opportunities and need for a holistic rethink of data protection systems.

Chapter 5

Model based Backup Scheduling

Data protection is one of the fundamental tasks in enterprise data management. Operational inefficiencies and scalability issues in data protection systems mainly stem from usage of static policy based management as highlighted in the data driven study in Chapter 4. In this chapter, we propose a model based dynamic backup scheduling framework that attempts to address key scalability and performance limitations of current backup systems. Specifically, we model backup performance as a function of both client and server side load characteristics along with configuration attributes and subsequently use these models for making smart backup scheduling decisions dynamically at runtime. We have developed a trace driven modeling & simulation framework for evaluating various operational and configuration optimizations that can aid in design of scalable backup ecosystems. Preliminary experimental results of model based scheduling using our trace driven simulation framework show that our approach can improve backup performance by as much as 32% for about 20% of backups and 13% improvement in average backup completion times.

Traditional data protection solutions follow a static hardwired model of clients, servers and backend devices where clients are installed with a backup agent that interacts with backup servers. Backup policies are set apriori with respect to policies and schedules without any adaptability to the backup workload variations. This hardwiring is part of initial configuration at provisioning time done by skilled domain experts. Such static paradigms lead to significant inefficiencies with improper resource utilization, performance hotspots for backup server(s) and eventually poor business continuity Service

Level agreements(SLAs) for modern enterprise.

Our proposed model based dynamic backup scheduling framework that attempts to address key scalability and performance limitations of current backup systems. We evaluate proposed dynamic scheduling using a trace driven simulation framework built to model real datacenter data protection environments. Rest of the chapter is organized as follows, - Section 5.1 gives an overview of performance variability in backup systems that can lead to significant operational inefficiencies, section 5.2 provides an overview of our proposed model based backup scheduling technique, section 5.3 discusses the trace driven what-if evaluation framework, section 5.4 provides experimental evaluation of proposed scheduling optimization and section 5.5 concludes with summary and future directions.

5.1 Performance Variability

Unpredictably long running backups, missed backups, backup window violations, partial backups are some of the top errors in backup environments. In order to better understand the key reasons for these class of errors, we analyzed performance characteristics of backups and we here we present a deep dive of the analysis for one selected backup server. Backup operations typically start around the same time every day as

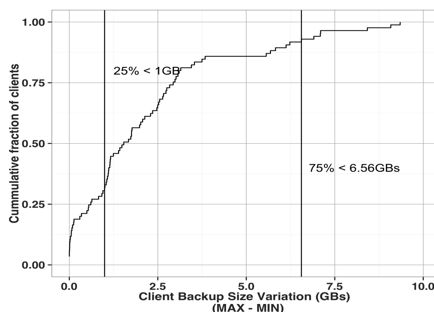


Figure 5.1: Variation in Size of Backups

per predefined static schedules. However, the completion time of these backups are a function of various parameters such as amount of data to backup, the server load, the client read bandwidth,etc. Figure 5.1 shows a cumulative distribution of variation in data size backed up on a daily basis over a 60 day analysis window grouped by client.

For about 25% of clients, the variation in amount of data backed up per day or the churn is less than 1 GB. For about 25% of the clients the churn is more than 6.5 GBs. Given these large variation in churn, one can expect significant variation in backup completion times. However, one would expect throughput of backups to be largely unaffected by these churn variations. Our analysis on the other hand shows that these clients exhibit significant variations in throughput as well. Figure 5.2 shows a cumulative distribution of variation in client backup throughput. For about 50% of backups, throughput variation ranges between 3.5 to 16 MBps. To illustrate the impact of such variation, consider an average system with 200GBs of data. Assuming a conservative change rate estimate of 15%, backup size would be 30GBs. Assuming a similar throughput range [3.5 - 16] Mbps, completion time could vary widely between [32 : 146] mins.

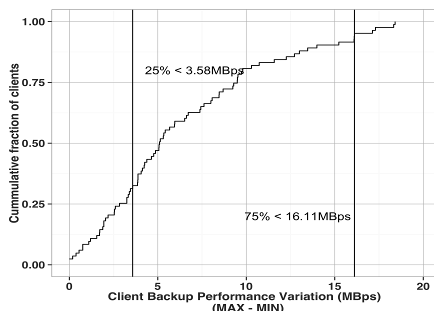
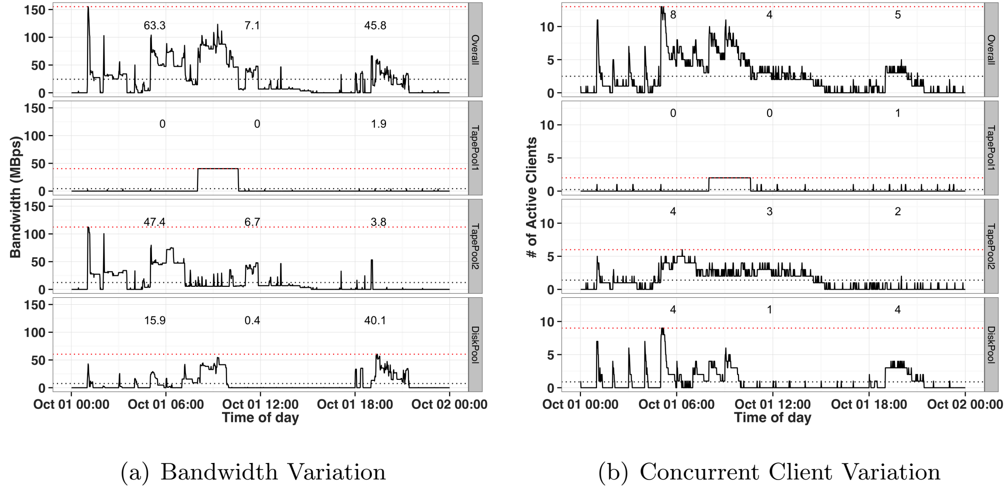


Figure 5.2: Variation in Backup Client Throughput

The reason for this variation in performance lies in variation in resource usage at the backup servers themselves. To better illustrate this behavior, we profiled performance utilization at the backup server on different days. Figures 5.3 & 5.4 show backup server Bandwidth utilization for a selected backup server on two different days, each plot showing a daily profile. Figures 5.3(a) & 5.4(a) show bandwidth variation on two different days both at an overall backup server level and for individual storage pools. Figures 5.3(b) & 5.4(b) show client activity variation on two different days both at an overall backup server level and for individual storage pools.

A high-level observation of bandwidth plots confirms static backup scheduling leading to consistent utilization patterns on backup server across different days. However, on closer observation, one can see the variation in bandwidth usage across different days



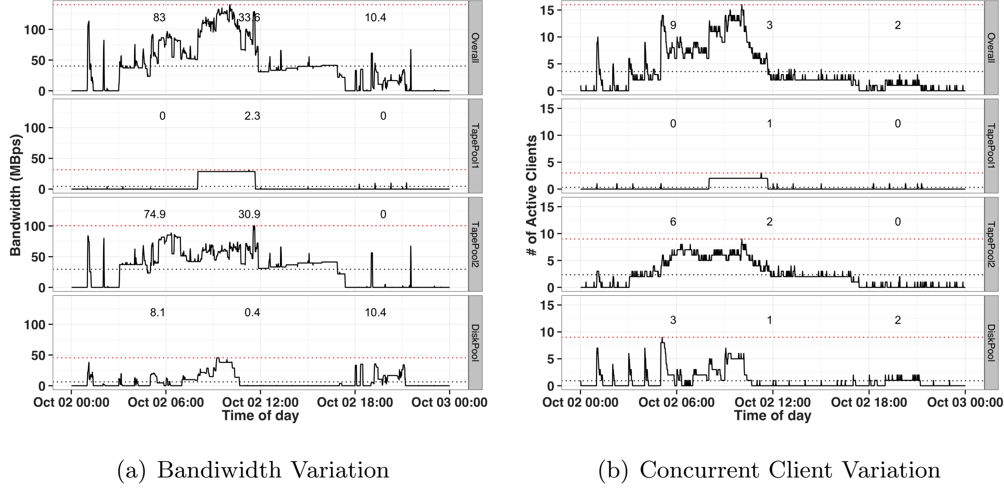
(a) Bandwidth Variation

(b) Concurrent Client Variation

Figure 5.3: Backup Server Load Variability (Day1)

for same time slices. For instance, overall bandwidth at 6am, noon and 8pm for day 1 are 63.3, 7.1 and 45.8 Mbps respectively, where as for day2, corresponding numbers are 83,33.6 and 10.4 Mbps. Clients which follow a static schedule - starting at same time daily would experience different behavior from the backup server as the server resource usage is significantly different. Further, such contentions are not just at the server level but also at the individual storage pool level. We believe that such variation in backup server side resource usage is one of the key reasons behind variation of backup performance/throughput observed by the clients. Cascaded effects of these variations are longer running backups and missed backups, backup window violations.

In summary, a chain reaction started by variation in daily churn/amount of data backed up coupled with static schedules leads to variation in resource utilization on backup servers - specifically leading backup servers into periods of overloads and conversely underutilization, resulting in variation in backup performance/throughput seen by the clients. The key ingredient facilitating the chain reaction is static scheduling and association policies that govern when backups start and where they backup to.



(a) Bandiwidth Variation

(b) Concurrent Client Variation

Figure 5.4: Backup Server Load Variability(Day2)

5.2 Backup Optimization

Scalability and performance limitations of current backup systems mainly stem from use of static policies for associations, schedules,etc.. As detailed in section 5.1, static nature of current data protection systems lead to suboptimal and inconsistent performance. Static policies often lead to contention of resources on backup servers. In this section, we describe a framework for optimizing backup performance using a model based approach.

Figure 5.5 shows an overview of proposed model based backup management framework. The framework constantly receives statistics on different backup jobs across multiple clients and also receives utilization/load information from each of the backup servers. Prediction models built using these live statistics and historical data are then used for determining optimization actions such as backup server selection, backup schedule selection and backup type selection dynamically during established backup windows. For the purpose of this study, we focus on using prediction models to determine when to initiate or start backups.

5.2.1 Model based Dynamic Scheduling

Objective of our framework is to maximize backup performance and to deliver consistent backup performance. To this goal, we designed a simple scheduling algorithm that

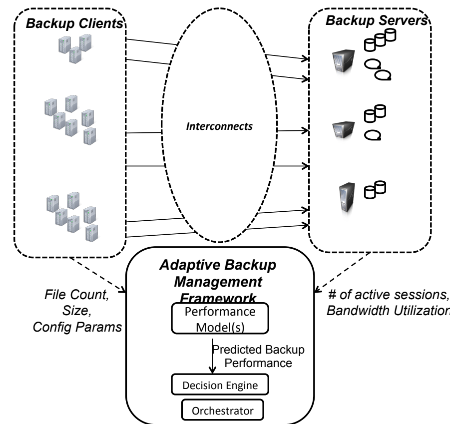


Figure 5.5: Adaptive Model based Backup Management Framework

exploits machine learning based models of backups to determine appropriate scheduling of backups.

Backup Performance Modeling

Backup performance is typically influenced by various quantitative and qualitative factors. Client side attributes such as the size of data to backup, the number of files in the backup and server side attributes such as number of concurrent backups on the server, bandwidth utilization on the server are the main quantitative factors. Qualitatively, the type of backup - full/incremental/differential,etc.., optimization enabled such as deduplication/compression are the main factors that influence backup performance.

Analyzing 60 day history of backups, using standard feature elimination techniques, we deduced a subset of features to build our prediction model. Top features selected include - size, type, file count, current backup server load parameters such as instantaneous and historical time average of overall bandwidth utilization, overall active client count, storage pool utilization and storage pool active client count. In order to limit interference, we created separate models of each client and backup server pair.

Further, we eliminated short lived backups (¡ 60 seconds) from the analysis as these short lived backups do not provide much room for optimization and are hander to predict. Using supervised learning approach we divided the dataset into 66% training and 33% test datasets respectively. After evaluating multiple regression techniques

such as Multi-Linear regression(MLR) and Support Vector regression(SVR) [76], we chose SVR as SVR prodded the best accuracies for our dataset.

Scheduling Algorithm

Typically each backup needs to be scheduled within a predefined window as agreed upon in SLAs. At every predefined time slice, our scheduling algorithm evaluates each backup request to determine its suitability for scheduling. Specifically, the algorithm uses the SVR model for the backup to predict expected execution/completion time given current server load conditions and backup requirements. All backups that are candidates at this time slice are sorted based on their predicted performance improvement. Improvement measured as % improvement in throughput compared to historical average. Top n of these backups are triggered and rest are queued for future consideration at subsequent time slices. n controls degree of parallelism and varies from system to system. For this study we set n based on our analysis of backup server active client activity to medium number of active clients for the server for the day.

One drawback of this approach is that the scheduling algorithm may not find certain backups suitable at any time slice with in their respective backup windows. To handle such scenarios, the scheduler implements a failsafe - at any analysis time slice, backup is trigger/scheduled even if predicted performance is below historical average if expected completion time based on average performance is beyond the backup window.

5.3 Trace driven Simulation

Understanding/evaluating the impact of optimizations via policy changes on real production deployments is often infeasible/impractical. To aid in such experimentation we build a trace driven discrete event simulation framework to mimic backup environments. Figure 5.6 shows a high level overview of our simulation framework. In order to match simulation environment to real production systems, we extracted configuration information from a production backup server and scheduling info for all its corresponding clients. Further, we extracted 60 day historical backup performance traces for all the clients and the load information across the same timeline from the backup server. Once the simulation models are configured according to the configuration traces, our trace

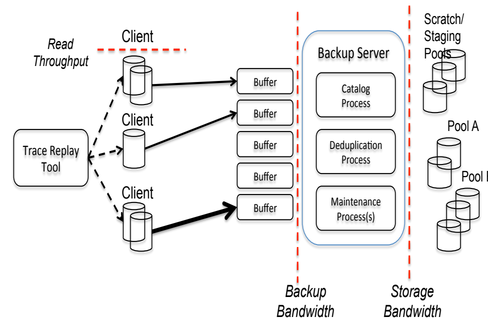


Figure 5.6: Trace driven Enterprise Backup Simulation Framework

replay tool generates backup requests based on timing and config information obtained from the traces. For purpose of this study, we replayed a 24hr trace on our simulation framework.

5.3.1 Calibration & Sensitivity Analysis

Based on aggregate analysis of all the traces across a 60 day period, we determined the peak bandwidth limits of backup servers and the underlying storage pools. We configure the models in the simulator with these identified bounds.In order to ensure correctness of our simulation framework, we performed two types of sensitivity analysis - by varying the backup bandwidth across the backup server and examining its corresponding impact on completion time of backups and consequently compare these times against actual observed values in production traces. About 10% of backups show a completion time mismatch of about 5%. For purposes of What If analysis this accuracy level is more than acceptable.

5.4 Experimental Evaluation

For the purposes of simulation & modeling, we limited our scope to uniform category of backup servers and incremental and full backups on these types of servers.

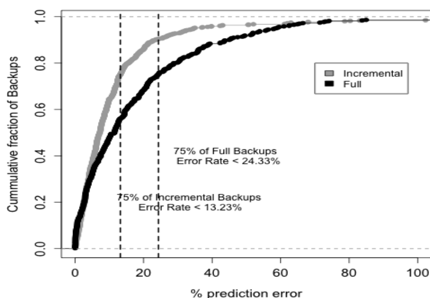


Figure 5.7: Modeling Accuracy for Backup Jobs

5.4.1 Modeling Accuracy

Figure 5.7 shows a cumulative distribution of prediction accuracies for two categories of backups - Full and Incremental. For the former, SVR based models were able to predict backup completion times with an error rate of less than 24% for over 75% of backups. For the latter, SVR based models were able to predict backup completion times with an error rate of less than 14% for over 75% of backups. On further investigation we found that the reason for this variation in accuracy across backup types was mainly due to the average duration/length of these types of backups. Median duration of full backups in our analysis was 190s, where as median for incremental was close to 800s. Further, in the environment analyzed, we found an order of magnitude more incremental backups than full. Given the magnitude of variation in backup performance, the accuracies of these models reasonable for use in optimization frameworks.

5.4.2 Scheduling Performance

In order to explore effectiveness of model based dynamic scheduling, we selected a subset of backups that exhibited high historical performance variation for schedule optimization. Figure 5.8 shows improvement in backup performance achieved using our proposed model based optimization framework. The x-axis shows the different clients (client_ids) selected for optimization and y-axis shows corresponding %age improvement in backup completion time. For about 20% of backups, performance improvement is greater than 32%. On an average, improvement in backup completion time is about 13%. However,

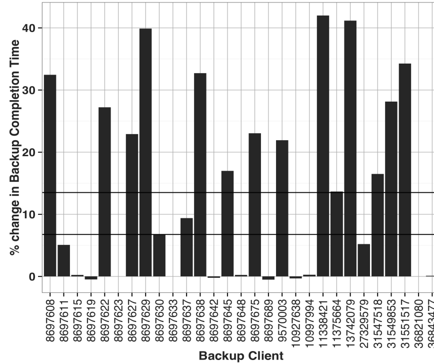


Figure 5.8: Improvement in Backup Performance with Model based Scheduling

for about 10% of backups, backup performance actually dropped, with worst case slow down of 5%. Further analysis showed that for these backups, our prediction models could not find any time slot with in their backup window where predicted performance was better than average historical performance. Hence, such backups got scheduled at the flag end of their backup windows leading to slight performance degradation.

5.5 Conclusions & Future Work

Data protection scalability and dynamic adaptability are key to building next generation data platforms. Our proposed model based scheduling framework takes into account instantaneous server and storage load explicitly and rearranges backup start times to optimize back performance. Preliminary results show that for about 20% of backups, performance improvement is greater than 32%. On an average, improvement in backup completion time is about 13%. A work in progress version of this work can be found in [77].

In this work we explored making on aspect of backup policies dynamic - the start time of backups. Other dimensions which we intend to explore as part of future work are client to storage and client to backup server associations. Use of trace driven framework allows exploration of various additional What-If scenarios. As future work, we intend to explore storage configurations of backup servers. Deduplication, tiering and replication on backup servers interact in complex fashion and provide significant room

for optimization in a cloud environment.

Chapter 6

Cloud object based Continuous Data Protection

Continuous Data Protection (CDP) enables recoverability to any point in time (time travel) facilitated via journaling of every write made by a system to disk. Stringent storage performance and capacity requirements for journaling make CDP a very high cost solution leading to limited adoption. In this chapter we first explore the feasibility of building such a CDP function on top of cheap commodity storage exposed via cloud object stores. Based on this analysis, we propose cCDP - a Cloud Continuous Data Protection framework that efficiently combines cloud object stores with edge caching to address requirements of low cost, high capacity, low latency and high storage throughput. cCDP with careful tuning can not only meet but surpasses the write throughput and latency requirements of CDP with minimal buffer overheads ($< 1\%$).

Cloud computing [78, 79] and Big Data [80, 81] have transformed IT from being a mere enabler, to becoming the centerpiece of the data-driven business differentiator for an enterprise. While agility and continuous integration of applications have been the poster children of these transformations, the IT administrators are now increasingly exposed to the nightmare of data corruption in business critical applications due to software bugs, hardware errors (such as bit rot), and operator mistakes with the rapidly changing management processes. Traditional backup-recovery while useful, is too coarse

grained (e.g. daily). Enterprises are increasingly demanding Continuous Data Protection (CDP) – recoverability to any point in time by continuously tracking all the updates made by the application.

From a systems standpoint, CDP solutions have been traditionally built using large pools of high speed storage in order to deliver acceptable write capacity and performance [44, 82]. Ideally, storage backends for CDP should match the write performance of storage used for the primary data and should be sized an order of magnitude higher than primary data as CDP not only stores the current latest version of every file or block but also all their recent complete histories. These stringent storage performance and capacity requirements have traditionally made CDP a very high cost solution, and hence its use has been limited to a very few high end business critical systems.

Cloud object stores are gaining significant traction in recent times as they offer a significantly more cost effective and pay as you go storage alternative to traditional on-premise storage alternatives. Commodity storage with scale out architectures coupled with economies of scale have made cloud object stores a viable offering for many cloud infrastructure providers [65, 66, 83, 84]. Cost models of these cloud object stores are driving increased integration into traditional on-premise storage stacks. Hybrid cloud infrastructures with tighter cloud object storage integration are gaining increased traction [85].

Though these cloud object storage alternatives can match traditional on-premise storage with respect to throughput, they fall short significantly on latencies. Average cloud object storage latencies are typically 3-5X higher than their on-premise block/file storage counterparts. To address these limitations, Cloud storage providers are now offering different variants of edge caching based on geo-locality (reverse Content Delivery Networks - CDNs) as value adds and sometimes as a key service differentiator [66]. Pricing models for these caching services vary from vanilla cloud object storage services.

cCDP is a framework for CDP using Object Storage backend, implementing innovative algorithms for de-staging into object storage, and data layout within blobs for fast retrieval. Specifically, this chapter explores a model of cloud object stores with efficient use of edge caching for building a CDP system with requirements of high capacity, low latency and high storage throughput. Given the objectives, we have selected open source OpenStack Swift object storage [64] as the low cost storage infrastructure due

to its highly available, distributed, and scalable architecture. Swift has seen increased adoption both in public storage clouds and in on-premise private clouds. The key contributions of this work are: a) feasibility study aimed at investigating components of building a CDP system using low cost, scalable cloud object storage. b) cCDP architecture - a framework that exploits Cloud object storage with edge caching to build a cost effective, horizontally scalable, high throughput CDP stack.

Rest of the chapter is organized as follows, - Section 6.1 highlights the performance limitations that arise in naive mapping of CDP workloads to objects. Section 6.2 provides a detailed design overview of our proposed cCDP system, Section 6.3 highlights key experimental evaluation results, Section 6.4 concludes with summary and future directions.

6.1 Feasibility Analysis

To understand the feasibility and performance intrinsics of CDP workloads and their suitability to object stores, we first implement the following intuitive method: every file or directory operation that needs to be logged maps to a new object. The data section of the new object holds both the metadata journal and the changed data. The metadata of the object itself is used to hold pointers to these two sections.

Figure 6.1 shows the test configuration where in every data or metadata write/delete operation to primary data is synchronously written to the swift object store as a new object. In a test virtual machine we configure filebench [86], a filesystem benchmarking tool to benchmark a block storage volume formatted with ext3 filesystem. Using FUSE [87] we integrate a CDP logger which works as follows : for every file or directory create/delete/write/append and metadata update operation on the ext3 formatted filesystem, the CDP logger performs two tasks in parallel labeled as 3a and 3b in Figure 6.1. Task 3a redirects control back to the underlying ext3 filesystem and invokes the required operations and task 3b creates and writes one or more objects with CDP required information as described earlier and stores them on swift object storage cluster. For example, consider a flush operation on a 1MB file. Based on filesystem block size (64 KB), the filesystem breaks down this flush into multiple block operations and the CDP logger maps each of these operations to a new object (64KB). As shown in Figure 6.1,

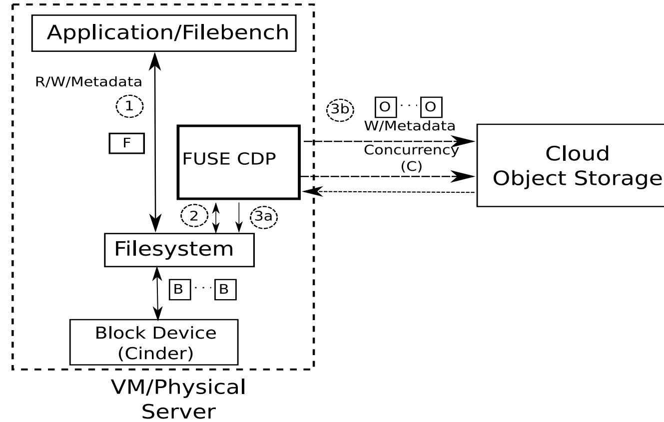


Figure 6.1: Naive File/Block Update to Object CDP

task 3b, these object write requests can be done concurrently (C) to exploit parallelism. Response or acknowledgment is sent back to the calling application only after both 3a and 3b are complete. Reads are not logged and are directly passed on to the underlying filesystem.

6.1.1 Ingestion Behavior

Table 6.1 shows performance comparison for two different benchmark workloads: fileserver and copyfiles on this testbed. The fileserver workload emulates a typical fileserver, which operates on a fileset of 20K, with a mean directory width of 20, mean file size of 1MB with a concurrency of 50 threads. Copyfiles workload emulates a typical directory copy operation - operates on a fileset of 10K, with a mean directory width of 20, mean file size of 1MB and operates in the context of a single thread. Baseline write throughput

Table 6.1: Impact of Synchronous Naive File-Object CDP

Workload Type	Baseline Write Throughput (Mbps)	Write Throughput with CDP(Mbps)				
		C=1	C=4	C=16	C=64	C=512
fileserver	142.3	24.1	26	25.7	**26.8**	24.7
copyfiles	87.3	2.7	5.8	7.7	9.3	**11.4**

achieved using fileserver workload (with no CDP logging) is about 142 MBps. Comparing the baseline with CDP enabled operation, we can see a write throughput degradation

of almost 19 times. Similar degradation can be seen in case of copyfiles workload as well - 13 times worse than baseline. Note that increasing concurrency/parallelism of object writes (during flush) shows non negligible improvement in case of copyfiles workload but not for fileserver workload. This behavior can be attributed to the fact that the workload itself - fileserver is already parallelized and metadata operation do not benefit from parallelism and hence does not see much benefit from object write parallelism (during flush). Each Metadata operation has to be written synchronously leading to significant overhead per such operation.

Drastic write throughput degradation evident from these results show that this simple mapping approach does not result in a usable solution. However, one of the main features of most cloud object stores is high write throughput. In order to understand the reason for this performance mismatch, we need to examine the object store write throughput characteristics. Figure 6.2 shows the write throughput variation of object store with varying object sizes and varying concurrency. In general, larger the objects, the higher the write throughput and higher the concurrency, higher the write throughput. To match the baseline write throughputs observed for non CDP enabled system (142 and 87 Mbps), the optimal operating point for object stores is at the very minimum (1M objects with concurrency of 32 or 2M objects with lower concurrency of 8). Further, the latencies of object operations are significantly different compared to that of block operations further contributing to the mismatch. Clearly, this *mismatch caused by naive mapping approach, specifically unbuffered direct mapping of filesystem updates which typically happen at granularity of file system blocks to objects leads to highly sub-optimal and unusable solution.*

6.1.2 Restore/Recovery Behavior

Typical CDP systems are designed for two types of use cases - a) Audit trails, intrusion detection and b) Disaster recovery and point in time recovery for testing. To support these use cases, a CDP system needs to have efficient mechanisms for a) identifying and ordering of objects to restore (for both use cases), b) copying over required objects and replaying the journal records to create required system state (for the latter use case). Both the *Identification & Ordering Time* and *Replay Time* can vary significantly based

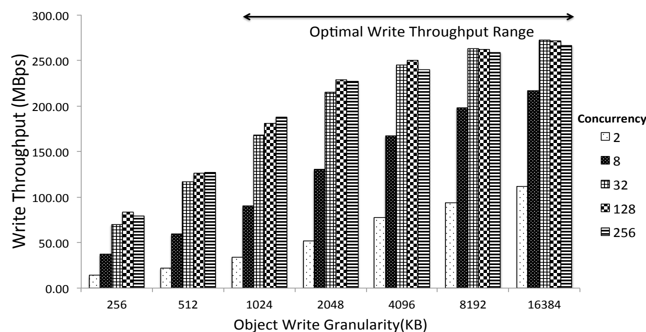


Figure 6.2: Performance Characteristics of Cloud Object Storage Cluster

on design choices.

Based on the simple mapping approach, identification and ordering of data to restore requires an exhaustive listing of all objects in the cloud object store, followed by get/read of all operations journal entries, followed by a filtering step based on user specified time range, followed by a sorting step. This full listing and sorting is necessitated by lack of built-in ordering support in object storage frameworks (traditional CDP systems rely on log structured approaches to achieve ordering). With the simple mapping approach, this listing and reading of all journal entries could potentially result in reading of billions of small objects significantly impacting data identification for restore. Further if careful parallelization is not done, actual replay of data can also be significantly impacted as a result of GETs on small data objects. Clearly, simple mapping approach *without additional indexing results in a highly suboptimal identification, ordering and replaying of data.*

6.2 cCDP System Design

Addressing CDP performance requirements, e.g., write and restore lookup performance, requires careful considerations and end-to-end optimizations of the entire IO stack. In the following, we first provide an overview of the proposed CDP system, and then discuss our layout management in details which can significantly impact on both the write and read data performance.

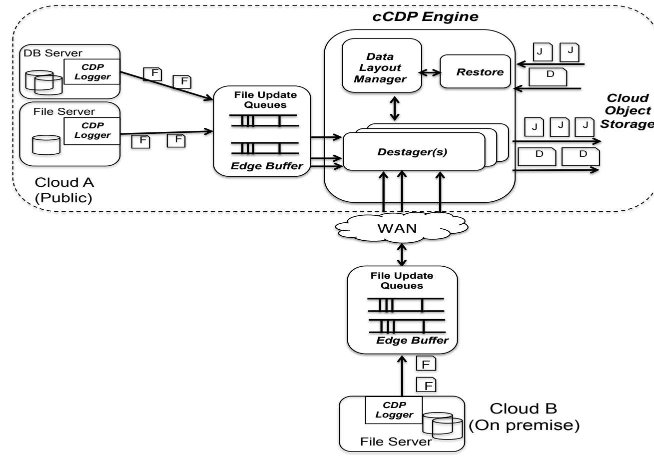


Figure 6.3: cCDP System Overview

6.2.1 Overall Architecture

Figure 6.3 shows the overview of the proposed CDP system. From a software perspective, the framework includes a client side module, namely, the CDP logger, which resides on the client systems and subscribes to the CDP service. An edge buffer cache close to the source system acts as a staging area and hides the network latencies of cloud access from foreground applications. The cloud service is composed of destager and restore modules that store and retrieve data from cloud object stores, respectively. The data layout manager acts as a mapper between filesystem updates and objects, which is a common component between the destage and the restore optimizers.

Based on the performance analysis highlighted in Section 6.1, latencies differ significantly across block and object storage systems. Even within the same datacenter, object access latencies are typically 3-4X worse than their block storage counterparts. Hence our proposed system incorporates an edge buffer cache that acts as a staging area to alleviate these performance limitations. The design rationale is not only to mask the performance characteristics of cloud object stores from the primary application, but also to reshape the write workload for better exploitation of cloud object stores via the staging area. Since write latency and throughput are the primary requirements for the edge buffers, our system uses a set of queues backed by an in-memory key-value store - Redis [88]. For each filesystem operation that needs to be logged (writes to

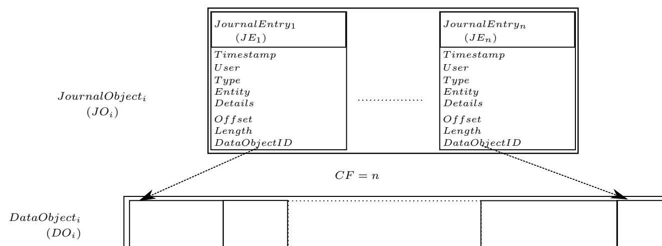


Figure 6.4: Object Structure: Journal & Data Associations

data and metadata), the CDP logger module in FUSE performs the following tasks - a) records the timestamp of this operation for ordering and consistency, b) dynamically picks a queue at runtime based on a simple round robin load balancing strategy and sends back an acknowledgement to the application. Each of these queues is served by a dedicated destage worker process which dequeues updates from the queue and performs the necessary packaging and writes the packaged object(s) to the cloud object store. Multi-queues with dedicated workers coupled with in-band timestamping of updates enable us to achieve a high degree of concurrency without any significant synchronization overhead.

6.2.2 Destager

Destaging involves dequeuing the operations from the edge buffers, grouping and mapping these file update operations to objects, instructed by the Data Layout Manager, and writing objects to swift. Each filesystem operation that the CDP logger needs to log consists of two types of data - an metadata journal and an optional data journal. From an access pattern perspective, the metadata journal are the primary data structures used for write placement decision-making and read/restore data identification and hence are more latency sensitive. In contrast, data journal are larger in size and require high read and write throughput and are less latency sensitive compared to operations journal records.

Our proposed system packs metadata journal and data journal into separate objects as shown in Figure 6.4. Each journal entry (JE_i) corresponds to a single atomic file system operation and consists of two types of metadata - a) metadata describing the filesystem operation, b) internal layout metadata which points to specific data object

(DO_i) and an offset within the data object that corresponds to the journal entry. Multiple journal entries and corresponding data are combined/coalesced together and stored as larger journal objects and data objects, in order to avoid creating a large number of small objects and to operate in a more bandwidth efficient region of swift.

The degree of combination or Coalesce Factor (CF) can theoretically equal to the maximum object size supported by the underlying cloud object storage framework (5GB in case of swift [64]), but high CF values have three important side effects : a) requires buffering all incoming updates until the CF is reached, which may lead to a higher buffer overhead, b) deletions in CDP system become inefficient since delete operations could require removing certain specific subset of journal entries from each object. Due to the limitations of the underlying cloud object storage framework, no update in place/partial writes is allowed in an object and the only option is to read the full object and then modify in memory followed by writing back the updated object, c) can potentially lead to less efficient bandwidth usage as primary copy of each object in swift is stored on a single storage server and possibly on a single disk without striping. For a given workload and object storage cluster, the choice of CF depends on the available buffer capacity.

Destager attempts to optimize swift write performance by coalescing operations. However, the determination of what operations to coalesce has a significant impact on read/restore performance, as will be discussed in the following section 7.2. From a throughput optimization perspective, our proposed system relies on encoding ordering semantics (timestamps) in the metadata and hence supports parallel destage workers with minimal synchronization overheads among them. One can scale the number of destage workers in accordance with their swift cluster performance capabilities to achieve optimal write performance.

6.2.3 Ordering & Error Recovery

Ordering guarantees are core to very CDP system. Traditional CDP systems mainly rely on log structured write semantics to ensure write ordering. Object storage systems do not provide any inbuilt ordering across objects. In our proposed CDP system, we enforce ordering by augmenting metadata of each operation with timing information at the source - in the CDP logger module on source system and moving the onus onto the read/restore process to base their processing on this augmented metadata. Added

benefit of this design is that all the intermediary processes can be designed agnostic of these strict ordering requirements. Specifically, the destager workers can be designed for write throughput optimization with little synchronization overheads.

By enforcing atomicity of operations across component boundaries and sequenced retries, our proposed system implements error recovery. Specifically, in the CDP logger module, a write operation is treated as successful only after completion of write on both the original filesystem and the edge buffer. Destager module removes queued requests from edge buffer only after successfully consistency checkpoints are reached. Checkpointing checks for completion of writes to swift and compares them to queued requests and only deletes queued requests that are temporally lower than highest successfully completed swift writes. An error recovery process scans the long queued unacknowledged requests and initiates a retry of such requests.

In-memory buffering of updates can pose a significant risk of data loss. In our design we rely on guaranteed durability of Edge buffers. Providers of such buffering services can achieve durability using various techniques. For instance, with Redis, one can achieve durability by using Clustered Redis or using Redis with battery-backed memory (NVRAM).

6.3 Performance Evaluation

6.3.1 Experimental Setup

CDP logger module is implemented using FUSE [87] bindings in Go language [89]. The CDP logger logs all updates to edge buffer in-memory queues. Edge buffer itself is implemented using list/queue abstractions provided by Redis [88], an in-memory key-value store. Destagers dequeue from Redis, applies necessary transformations by consulting Data Layout Manager and writes objects to an openstack swift object storage cluster made up of 2 proxy servers, each with 10 GB eternal links and 6 storage servers each with local attached SCSI disks and 4 GB network links.

To evaluate of proposed system, we make use of filebench, an open source filesystem benchmark [86]. Filebench supports different load profiles or personalities such as web server, fileserver, video server,etc. For our study we choose the fileserver profile as it is very representative of typical high volume enterprise storage servers. Specifically, we

configure fileserver profile with an initial fileset of 100K files, with a mean directory width of 20, mean file size of 1MB with a concurrency of 50 threads. All performance results shown henceforth are with filebench benchmark using this fileserver personality. Each benchmark was run for a duration of 600 seconds and repeated thrice to eliminate statistical variations. At the start of each test, Filebench first creates a fileset of 100k files - about 100GB in total size and then we run the fileserver workload profile and our measurement interval starts from this point onwards for a duration of 600 seconds. During the filebench IO activity period post initial creation, filebench generates about 45GBs of write File IO that is a mix of data and metadata modification operations. Since foreground filebench workload synchronously writes to in-memory queues backed by Redis, the latency impact on foreground workload is minimal (writing to local disk vs writing to Redis memory queues). Hence, we focus our evaluation on: a) the Edge buffer usage overhead incurred in transforming the workload from primary filebench file system updates to object friendly workload, b) the impact of data layouts on write throughput in the form of Edge buffer usage and data amplification due to indexing , c) the impact of data layout on Restore data identification (since Replay Time is constant across the different data layout alternatives)

6.3.2 Results

Ingestion Performance

Edge buffers play a crucial role in transforming the file CDP workload into object friendly workload. Edge buffers are either client provisioned services or are typically value added services offered on top of vanilla cloud object storage services by cloud providers to assist latency sensitive operations. Given the in-memory nature, they are typically an order of magnitude more expensive than at-rest cloud object stores. Our proposed system helps to optimize the usage of these buffers. Figure 6.5 shows the variation in edge buffer usage as a function of Coalesce Factor (CF). In general the lower the CF and concurrency (parallel IO), the higher the edge buffers required. For a low CF value 32, with corresponding low concurrency 8, the peak buffer usage needed for the filebench workload is quite high - 26 GBs (with a mean usage of about 13GBs). Similar trend can be seen for other lower CF of 64. A closer look at the statistics of objects produced by these transformations indicates that the average object size

produced by CFs 32,64 are about 0.8 and 1.6 MBs respectively. At this object size and concurrency we are not operating in Swift's optimal throughput range and hence the exaggerated buffer usage.

Higher CF and Concurrency can provide better buffer usage (lower) but after a certain point, the performance actually degrades. For instance with higher concurrencies of 32 and 64 and higher CFs of 256 and 512 we observe that the peak buffer usage actually increases. Main reasons behind this issue are - a) swift cluster becomes a bottleneck, b) destagers wait until their CF is satisfied before attempting a destage to swift. Hence, for the workload under test and our object storage cluster setup, optimal CF value is 256 with a concurrency of 16. Table 6.2 shows the statistics of swift objects created for different values of CF. For a CF value of 256, average size of swift object created is about 5.9 MBs confirming that our transformation operates in an optimal write throughput range for the swift cluster.

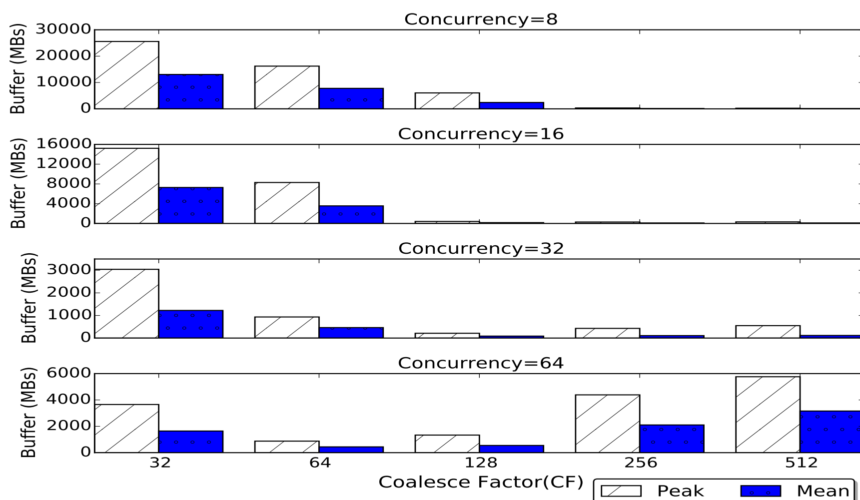


Figure 6.5: Impact of Coalesce Factor on Edge Buffer Usage

6.4 Conclusions & Future Work

CDP enables recoverability to any point in time and caters to a wide variety of data protection requirements. Through detailed benchmarking, experimental analysis and

Table 6.2: Impact of Coalesce Factor (CF)

CF	# of operations logged	# of Objects	Mean Object Size(KBs)	Total Size(GBs)
32	2099345	63631	753.14	46.8
64	2045429	27688	1690.04	45.4
128	1980081	15347	3009.2	45.1
256	1993680	7754	5903.12	44.7
512	2015526	3925	12314.08	47.2

prototyping we identified challenges in building a CDP system using low cost cloud object storage built using commodity hardware. Our proposed cCDP system design incorporates edge buffering and destage optimizations that results in CDP write throughputs comparable to raw block based enterprise storage with very minimal memory requirements ($< 1\%$) with appropriate choice of Coalesce Factor (CF) and concurrency. For additional details of this work please refer [90].

Future work involves dynamically optimizing Coalesce Factor (CF) and concurrency level based on observed workload behavior and performance of underlying object storage cluster, impact of edge buffer physical placement on overall CDP design, detailed exploration of deletion and retention requirements and how they map to object lifecycle, potential enhancements to cloud object storage frameworks to better aid data protection workloads, application of techniques explored in this paper to other cloud object stores.

Chapter 7

Recovery Optimizations for Cloud Continuous Data Protection

Data protection offered by Continuous Data Protection (CDP) solutions is significantly richer when compared to traditional data protection solutions. Operational efficiency and usability of CDP is however a function of how efficiently data can be restored in case of a failure. Specifically, ability to quickly search thru CDP logs to identify data that needs to be restored and subsequently fast tracking the restore data transfer operation are key to CDP. In this chapter, we propose a) a novel method of organizing layout of CDP logs on object storage to exploit object storage retrieval characteristics and optimize temporal search performance, b) an object naming encoding scheme coupled with a Trie based queueing mechanism to optimize spatial search performance of CDP based restores. Preliminary results show that Temporal search performance of cCDP with proposed hybrid data layout is within 28% of Btree based temporal data layout and is about 52.7% better than the naive data layout with name encoding with significantly lower edge buffer and write throughput overheads than the Btree based temporal data layout. Further, Spatial search performance of proposed object name encoding and Trie based queueing is about 15X more efficient than naive encoding and 12X faster than naive encoding.

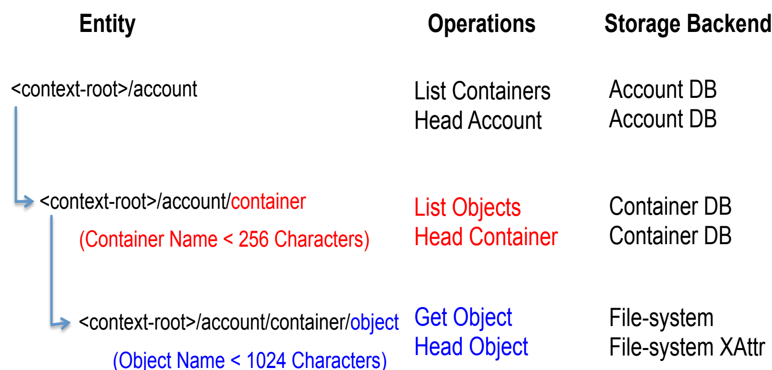


Figure 7.1: Retrieval Operations supported by Cloud Object Storage(Swift)

Recovery operations in CDP can be broken down into two phases, namely - a search phase and a restore phase. The search phase involves identifying the relevant set of updates from CDP logs to restore. Based on the use case, full system or individual file recovery, this search phase requires either just a temporal search or a combination of temporal and spatial search through CDP logs. Once the search phase completes, recovery phase is mainly centered around steaming performance similar to any other CDP implementation. Search phase however is quite unique to object storage systems. One needs to consider the retrieval characteristics of object storage system to optimize search behavior. Section 7.1 provides an overview of object storage retrieval characteristics. Section 7.2 details how data layout can be exploited to optimize search behavior. Section 7.3 details how name encoding can be exploiting in object storage systems to optimize for search performance. Section 7.4 provides preliminary results for data layout and naming optimizations. Section 7.5 concludes with summary and future directions.

7.1 Object Storage Retrieval Characteristics

Retrieval operations supported by cloud object storage along with the hierarchy at which they operate are shown in figure 7.1. In total, three main types of retrieval operations are supported by object storage system, -

1. *Head Operation:* is supported at all levels of hierarchy. At an account level, head operation retrieves mainly account metadata attributes such as usage info, access

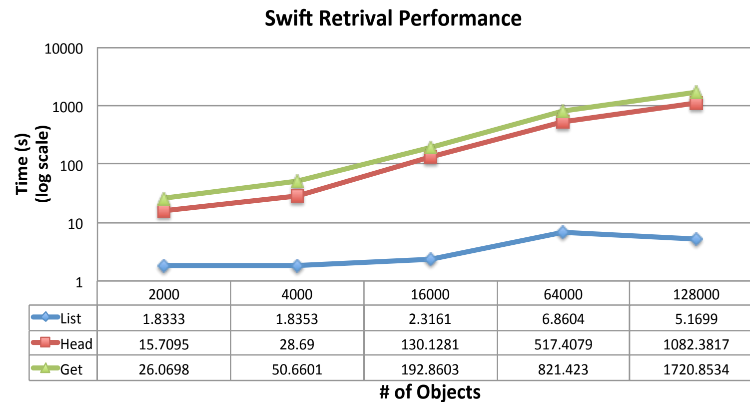
control. In swift, this operation is implemented using a separate database that maintains account level metadata resulting in very fast access performance. At a container level, head operation retrieves container metadata attributes such as policies and quota usage information. In swift, this operation is implemented using a separate database that maintains container level metadata resulting in very fast access performance. At an object level, head operation retrieves user settable metadata attributes. In swift, this operation is implemented using file system extended attributes maintained by underlying filesystem resulting in slightly slower access performance comparable to random file access.

2. *List Operation:* is supported at account and container level. At an account level, list operation lists names of all containers held within the account. In swift, this operation is implemented using a separate database that maintains account to container mapping resulting in very fast access performance. Each list call returns a maximum of 10000 entries and additional entries can be retrieved by paginating successively. Similarly, at a container level, list operation lists names of all objects held within a container. In swift, this operation is implemented using a separate database that maintains container to object mapping resulting in very fast access performance. Each list call returns a maximum of 10000 entries and additional entries can be retrieved by paginating successively. Further, list operation also supports prefix based filtering of entities.

3. *Get Operation:* is supported at object level. Get operation results in retrieval of either entire object or a byte range within an object. Access performance is slower compared to list and head operations as data needs to be fetched from underlying filesystem on object servers and is a function of file of the object.

Naming convention supported in swift limit the container name to 256 characters and object name to 1024 characters.

Figure 7.2 shows Retrieval characteristics of a sample Swift object storage cluster comparing performance of list, head and get operations. Each point on the x-axis represents a specific number of entities used for benchmarking. For this characterization, we used a fixed object size of 8K. In general, list operations are an order of magnitude faster than either get or head operations. This shows the disparity in access performance



	2000	4000	16000	64000	128000
List	1.8333	1.8353	2.3161	6.8604	5.1699
Head	15.7095	28.69	130.1281	517.4079	1082.3817
Get	26.0698	50.6601	192.8603	821.423	1720.8534

Figure 7.2: Retrieval Performance Characteristics of Object Storage (Swift): List vs Head vs Get

between meta data stored in a database as appose to data distributed on underlying filesystem. Further, head operations are slightly faster than the get operations. With larger object sizes, one should expect this difference to increase.

From the conceptual view shown in Figure 7.1, and experimental characterization shown in Figure 7.2, the research challenge is to design data organization and naming convention to exploit these object storage retrieval characteristics.

7.2 Data Layout Optimization for Full System Recovery

Physical layout of objects on swift can have a significant impact of restore/read performance and can also impact on write ingestion performance. Full system recovery mainly requires a temporal search in the CDP logs to determine the data to restore. In this section we highlight different data layout strategies and compare and contrast them based on their lookup efficiency and write overheads.

Figure 7.3 shows a naive layout with two swift containers - one container for all journal objects and another for all data objects.

From a read/restore perspective, finding the data based on temporal constraints, e.g., between a specific time range, requires the following steps - 1) list all journal objects in the journal container, 2) retrieve data of each journal object, 3) filter journal entries in each journal object based on specified time range. Step 1 can take a significant amount of

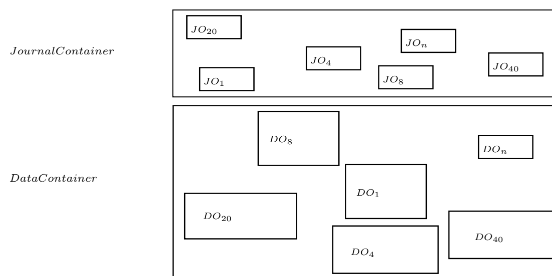


Figure 7.3: Naive Layout (1 Data Container, 1 Journal Container)

time based on number of journal objects in the system. Further, swift paginates such list requests to limit performance impact (with a typical paging of 10k entries). Irrespective of the length of the time range to restore/read, this simple layout necessitates a full listing and read of all journal objects. Over typical lifetime of cdp, journal container could have billions of objects and this simple layout can severely impact recovery time and is the least practical of the alternatives. One optimization on this simple layout is to encode the temporal metadata of journal object in its name. For instance, encoding lowest and highest timestamp of constituent journal entries as the name of the journal object can provide appreciable speedup. Step 2 now can include a pre-filtering step based on name of the journal object. From a write perspective, this simple layout has the least overhead on write performance. Journal and data objects can be written at wire speed to swift into their respective containers. Further, this naive layout does not lend itself to easy parallelization due to lack of ordering semantics for list operations and need for strict sorting.

Figure 7.4 shows a Btree based temporal layout with two swift containers - a container for all Btree index nodes and another for all data objects. Instead of packing journal entries into larger journal objects based on just size heuristics, in this layout, each journal entry is inserted into a Btree with the timestamp as the key and each Btree node itself is stored as a swift object in the journal container. From read/restore perspective, finding data to read/restore based on temporal constraints, i.e.., finding data between a specific time range requires a single step - a Btree range query - inorder tree traversal with temporal bounds. Based of the length of the time range to restore/read,

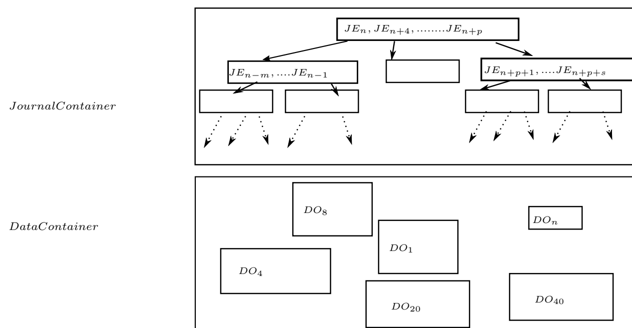


Figure 7.4: Btree based Temporally Indexed Layout (1 Data Container, 1 Journal Container)

this Btree layout can provide significant speedup compared to simple layout as it eliminates the need for full scans. From a write perspective, every insert of a journal entry into this index can potentially result in $\log_D N$ Btree node updates, where N is the total number of Btree nodes in Btree index and D is the degree of Btree (size of Btree node: $2D - 1$). However, object stores do not support the partial update of objects and hence the entire object or set of objects corresponding to affected Btree nodes needs to be rewritten - $\log_D N$ object writes/rewrites. For every journal entry insert, the write amplification is $O(\log_D N)$. One optimization on this Btree approach to minimize these rewrites is that we do not flush these corresponding objects on every journal entry insert but batch it for every data object which contains data corresponding to multiple file updates or journal entries based on CF chosen. This can reduce the write amplification and corresponding Btree node/swift object rewrite penalty is incurred once per larger data object write instead of each journal entry insert. Further, this Btree based temporal index facilitates easy parallelization of lookup queries. By dividing temporal range queries into multiple smaller range queries and executing them in parallel, this layout can achieve significant speedup over other approaches.

Figure 7.5 shows a hybrid layout with one container for index nodes, variable number of containers for journal objects and one container for data objects. In this layout, each journal object containing set of journal entries, is placed in appropriate journal container based on a query of a Btree index on containers with container name as the key. Container name is used to encode the lowest timestamp of journal objects contained in

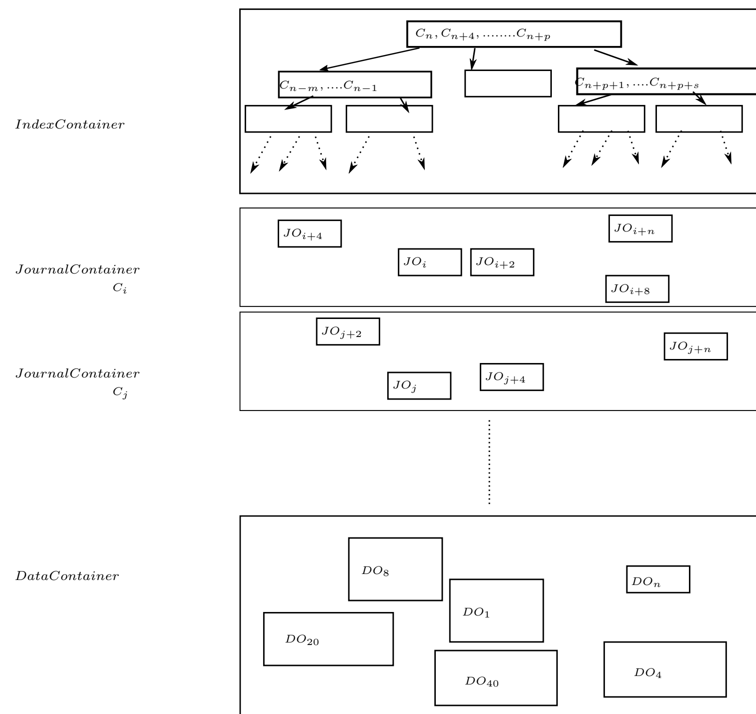


Figure 7.5: Hybrid Layout (1 Data Container, n Journal Containers, 1 Index Container)

it. Successive containers are responsible for disjoint time ranges. For instance, all journal objects/entries in container C_i within time range $[i, j)$, where j is lowest timestamp of next container C_j. From read/restore perspective, finding data to read/restore based on temporal constraints, i.e.., finding data between a specific time range requires the following steps - 1) a Btree range query - inorder tree traversal with temporal bounds to determine candidate containers, 2) retrieve data of each journal object in the candidate container, 3) filter journal entries in each journal object based on specified time range. Just as in the simple layout, one optimization on this layout is to encode the temporal metadata of journal object in its name. Step 2 now can include a pre-filtering step based on name of the journal object. From a write perspective, every new journal object requires a query of container Btree to determine the appropriate journal container. If resultant container has more than a predefined threshold number of journal objects, a new journal container is created and added to the container Btree and the journal object is written into this newly created journal container. By matching threshold with swift pagination behavior (10k objects), we can exploit ideal swift listing efficiencies. In practice, in order to accommodate for out of order writes of operations, the threshold should be set slightly lower than the pagination behavior. Write amplification due to Btree inserts and penalty due to Btree node/swift object rewrites is incurred once per new container creation instead of each journal entry or journal object write (container creation is several orders of magnitude rarer than journal entry or journal object creations). This hybrid layout combines the low write overheads of the naive layout with high restore/lookup performance of Btree based layout. Further, similar to Btree based layouts, this hybrid index layout also facilitates easy parallelization of lookup queries. By dividing temporal range queries into multiple smaller range queries and executing them in parallel, this layout can achieve significant speedup over other approaches. However, since within a container, there is no strict ordering, an additional local sorting step is required before merging with other containers to produce an ordered set of lookup results.

7.3 Naming optimizations for Individual File/Subtree Recovery

Individual File/Subtree Recovery requires both temporal and spatial search thru the cdp logs to determine data to be restored. Section 7.2 addressed the mainly temporal search. In this section our design goal is to efficiently support spatial search for individual file or subtree recovery.

7.3.1 Exploiting Naming Semantics

Figure 7.1 highlights different naming conventions supported by swift. Limit for the container name is 256 characters and object name is 1024 characters. Given these naming conventions and the performance characteristics of swift described in Section 7.1, our goal is to design a naming scheme to exploit the retrieval behavior of swift. Several alternatives exist for such name encoding, namely -

- Container name: Time range, Object name: longest common prefix of filepath + time range
- Container name: Longest common prefix of filepath, Object name: time range
- Container name: Longest common prefix of filepath, Object name: longest common prefix of filepath + time range
- Container name: longest common prefix of filepath + time range, Object name: longest common prefix of filepath + time range

Based on swift container naming and object naming limitations, the most flexible alternative for CDP encoding is container name for Time range encoding and object name for encoding a combination of longest common prefix of filepath and time range. With this encoding scheme, container naming helps prune the search space for temporal search and object naming helps prune the search space for spatial search. Further, this scheme helps minimize the number of get requests required for the overall search operation.

7.3.2 Semantic Coalescing

Spatial search can significantly benefit from object name encoding. With optimal encoding, spatial search performance can be significantly improved by exploiting list api calls intelligently and minimizing the number of expensive get api calls required for pruning search space. Recall from Figure 6.4 an object is composed of multiple filesystem updates coalesced together for purpose of creating large swift objects that are write throughput optimized. These updates that constitute an object can belong to multiple different files on the protected filesystem. The extent of interleaving of updates from multiple files is a function of the workload. Specifically, concurrency characteristics of applications dictate the interleaving. By using Longest Common Prefix (LCP) of filepaths amongst all constituent updates as the object name, significant information can be encoded in the object name that is retrievable by inexpensive list api calls. The more precise the LCP, the more stronger the encoding, i.e., easier pruning of search space. LCP depends significantly on how the incoming updates are grouped or coalesced together and is a function of workload. In the following subsections, we explore two different alternatives to optimize LCP - FIFO LCP queueing and Trie based LCP queueing.

FIFO based Longest Common Prefix Queuing

Figure 7.6 shows a First In First Out based LCP queueing. In this model, multiple concurrent application writers queue write requests on a common shared queue. A destage process dequeues requests asynchronously from this shared queue and adds the requests to a secondary queue when it detects space availability on the secondary queue. When number of updates on secondary queue becomes greater than or equal to optimal swift object size, a new object is created with these updates and written to swift object store. The name of the object is derived as the longest common prefix of filepaths of all constituent updates that make up the object. Advantage of this queueing system is that it ensures fairness - updates are processed and make way to swift in the order they were written and it is very simple to implement - double buffering is widely used technique. Downside of this approach is that if application concurrency is high, the common prefixes tend to be short and provide very little information gain. The object

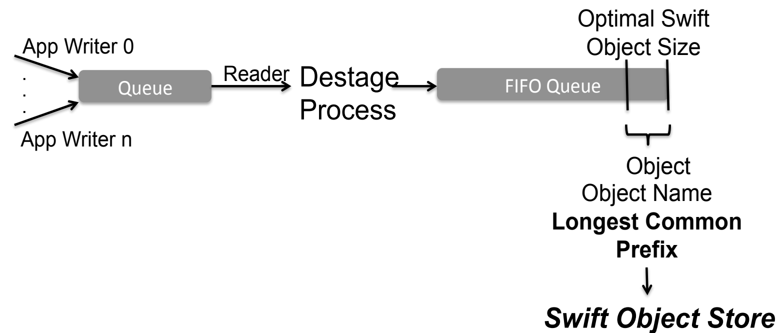


Figure 7.6: FIFO based Least Common Prefix

name ends up being quite generic.

Trie based Longest Common Prefix Queuing

Figure 7.7 shows a Trie based LCP queueing. In this model, similar to fifo based approach, multiple concurrent application writers queue write requests on a common shared queue. A destage process dequeues requests asynchronously from this shared queue and inserts the requests to a secondary buffer structure organized as a Trie [91] with filepath as the key when it detects space crunch on the primary queue or based on precofigured frequency. Insert operation on Trie works as follows, - a) filepath of request is used to locate appropriate tree node where the request needs to be queued. For instance, for a request with filepath /db/ts2, the insert checks if nodes /, db and ts2 exist in Trie. If not found, these nodes are added to Trie. b) Upon finding an appropriate node in Trie, the request is inserted into the buffer at this node.

Destaging from this Trie buffer to swift store works as an asynchronous process. Algorithm 2 shows the pseudo code for destaging. Destage algorithm works progressively down the the Trie structure as follows, a) If current node buffer usage is greater than or equal to optimal swift object size, one or more swift objects are created based on this buffer with relative trie path as the object name, b) If buffer usage of any child nodes of current node are less than threshold (swift optimal size), buffers of all such children are coalesced and one or more swift objects are created based on this combined buffer with relative trie path as the object name, c) If buffer usage of any child nodes of current node are greater than or equal to threshold (swift optimal size), the algorithm

recursively proceeds with one such child node as the next current node.

Algorithm 2 Trie Queue Destage Algorithm

```
 1: procedure FLUSH_TRIE
 2:     while (true) do
 3:         Set root of trie as start_node
 4:         nodes_to_flush ← DETERMINE_NODES_TO_FLUSH(start_node)
 5:         for each node_to_flush do
 6:             object_name ← node_to_flush.absolute_path
 7:             object_data ← node_to_flush.buffer
 8:             SWIFT.WRITE_OBJECT(object)
 9:             Clear node_to_flush from Trie
10:         end for
11:         SLEEP(flush_interval)
12:     end while
13: end procedure
14: procedure DETERMINE_NODES_TO_FLUSH(node)
15:     Initialize a list of flush_nodes
16:     if sizeof(node.buffer) > optimal_object_size then
17:         flush_nodes.append(node)
18:     end if
19:     for each child_node do
20:         subtree_buffers ← CHILD.GETALLSUBTREEBUFFERS
21:         if sizeof(subtree_buffers) > optimal_object_size then
22:             flush_nodes.appendAll(DETERMINE_NODES_TO_FLUSH(child_node))
23:         else
24:             child_node.buffer.append(subtree_buffers)
25:             flush_nodes.append(child_node)
26:             Clear child_node subtree
27:         end if
28:     end for
29:     return flush_nodes
30: end procedure
```

Advantage of this queueing system is that it ensures longer prefixes largely independent of application concurrency profile there by ensuring significant information gain. The object name ends up being specific. Downside of this approach is that fairness criteria is compromised. Updates can make their way to swift irrespective of the order in with application writes them. If preconfigured frequencies or *flush intervals* are used, larger the frequency (less time between destaging), lesser the prefix length and higher the fairness. Conversely, less frequent destaging results in longer prefixes but lesser degree of fairness.

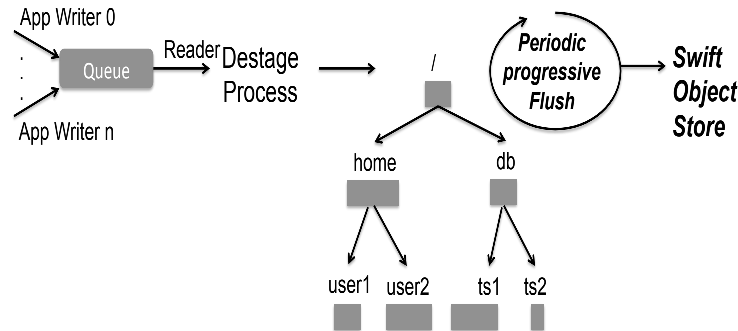


Figure 7.7: Trie based Least Common Prefix

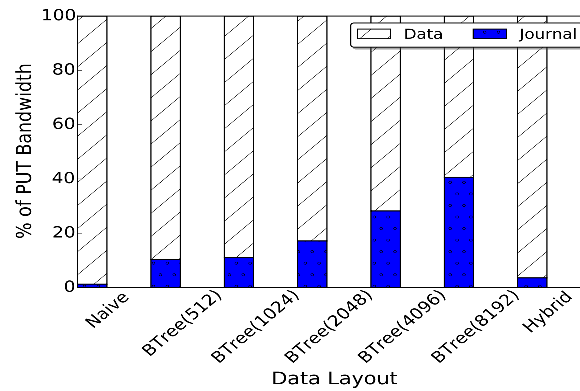


Figure 7.8: Impact of Data Layout on PUT performance

7.4 Performance Evaluation

7.4.1 Data Layout write overhead

Layout of data - journal objects and data objects has a significant impact on not only the restore or lookup times but also on ingestion performance. Figure 7.8 shows a breakdown of size of data and journal write bandwidth usage for different data layout schemes. Layout schemes for comparison are - Naive layout (one container each for data and journal objects with no indexing), Btree layouts with varying degree (one container for data and another for journal objects organized as a Btree) and our proposed Hybrid layout (one container for data and one container for container Btree nodes and variable number of containers for journal objects). For the naive layout, at the end of benchmark

Table 7.1: Impact of Data Layout on Journal Rewrites

Degree of Btree (D)	Mean number of Btree Node/Swift Object Rewrites
512	3.95
1024	5.89
2048	9.47
4096	17.47
8192	30.59

run, total size of data objects is about 44.2 GBs on swift and total size of journal objects is about 600 MBs. Journal entries account for about 1.2% of the total swift capacity usage. Since, naive layout relies on directly writing data and journal objects into their respective containers without any transformations, the percentage of swift PUT bandwidth usage is also 1.2 and 98.8% for journal entries and data respectively. However, for Btree based layouts, as the degree of Btree increases (flatter tree with larger nodes), the percentage of PUT bandwidth used for journal entries increases dramatically. For larger degree of 8192, the journal objects entries account for about 40.6% of total PUT traffic to Swift. Note is that the actual at rest size of journal entries is still constant. This is a direct impact of Write amplification (every insert of journal entry could potentially result in $\log_D N$ Btree dirty nodes that need to written to swift) and having to rewrite entire large Btree node on every insert of a journal entry as the platform does not support partial writes or range writes with in existing objects. For these experiments we batch journal entries belonging to larger data objects and batch commit the btree nodes along with corresponding data object. However, as shown in Table 7.1, the impact of write amplification and full rewrite of Btree nodes can be significant. For instance, for a Btree degree of 1024, mean number of Btree node (corresponding swift object) rewrites is about 5.89. This value increases significantly with increase in Btree degree - flatter trees, more common/dirty nodes. Our hybrid layout is very comparable to naive layout interms of percentage of write bandwidth consumed by journal objects. Overhead of the container Btree index is very minimal ($< 2\%$) given that container creation is several orders of magnitude rarer events than journal object creation.

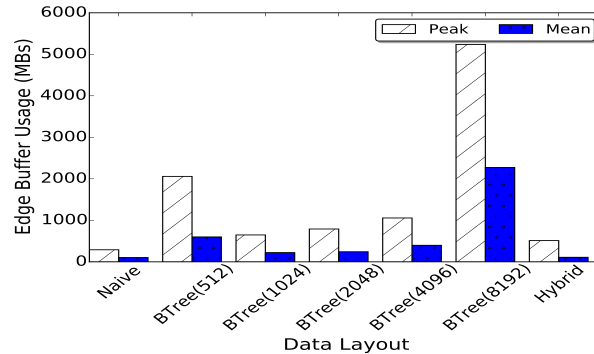


Figure 7.9: Impact of Data Layout on Edge Buffer Overhead

Figure 7.9 shows the edge buffer usage for different data layout schemes. For these comparisons we use the CF and concurrency values identified in Section 6.3.2 256 and 16 respectively. To recap, for naive layout, the mean and peak edge buffer usage is about 109 and 304 MBs respectively. For Btree based layouts, with a low degree of 512, the mean and peak edge buffer usages increase to about 600 MB and 2.15 GB respectively. However, for degrees 1024,2048 and 4096, the edge buffer usage is much lesser. The main reason behind this trend is that with smaller Btree nodes, inserts into the Btree can cause signifiant write amplification leading to lot of small dirty Btree nodes to flush. Conversely, for a higher degree of 8192, on every insert - a) less number of larger Btree nodes need to be flushed due to lack of partial updates of objects and b) with larger nodes, time completely of sorted insert within the node dominates. leading to higher edge buffer usage. Our hybrid layout leads to slightly higher memory usage of 115 and 536 MBs compared to naive layout. Main reason for the slight increase is the added complexity of determining the correct container placement for each journal entry. Overall, hybrid layout does not lead to significant edge buffer overheads unlike the Btree based layouts.

7.4.2 Temporal Restore Search Performance

Data layouts can have a profound impact on identification & ordering performance (IO_{time}) . Replay time however is constant across the different alternative. Hence, we focus on evaluation on IO_{time}. Figure 7.10 shows a comparison of Identification &

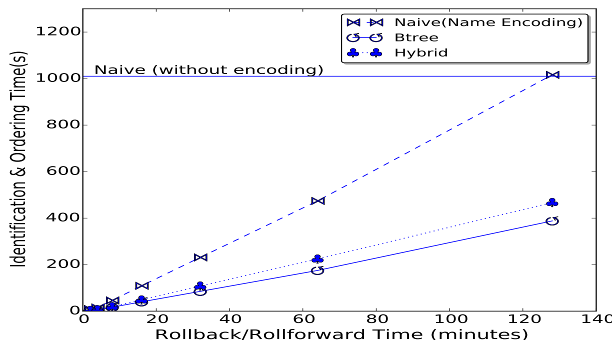


Figure 7.10: Temporal Search Performance: Identification & Ordering Time

Ordering performance (IO_{time}) for different data layout strategies. For this comparison we use a longer 2 hours segment of filebench run which generates about 442 GBs of data with about 20 Million journal entries. For naive layout, irrespective of rollforward/rollback point in time, the IO_{time} is about 1010 seconds. With the object name encoding heuristic (with lowest and highest journal entry timestamps), this behavior changes significantly and IO_{time} varies directly with rollforward/rollback point in time. For rollback/rollforward to nearly the half way point - 64 mins, naive layout with the name encoding takes about 474 seconds for identification and ordering of appropriate journal records (10.1 Million). Btree based layout we choose a Btree with degree $D = 1024$ based on edge buffer memory overheads highlighted in previous section. This layout provides the best IO_{time} performance as expected. Our proposed hybrid approach follows the performance of Btree based layout quite closely. For rollback/rollforward to 64 minutes, hybrid approach is within 28% of Btree based approach and is about 52.7% better than the naive approach with name encoding. Both Btree based and our propose hybrid index based layout show significant speedup over naive approach in part due to parallelization of lookups. Clearly, the hybrid approach provides a reasonable IO_{time} performance within range of Btree based approach with significantly lower edge buffer and write throughput overheads.

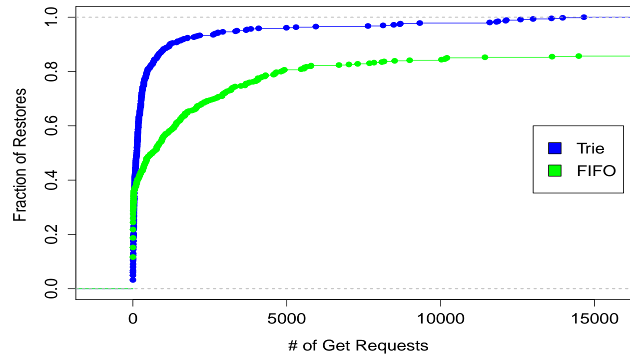


Figure 7.11: Comparison of FIFO and Trie LCP Queuing

Table 7.2: Impact of Queueing on Restore spatial search

Api Request Type	FIFO LCP	Trie LCP
List	21.76	7.2
Get	10999.73	727.5
Useful Get	57.89	67.52
Time	210.58	17.83

7.4.3 Spatial Restore Search Performance

Name encoding and Queueing impact Spatial search performance. Figure 7.11 shows a comparison of restore spatial search performance between FIFO based queueing and Trie based queueing. For this experiment, one file of each access pattern type of chosen randomly for restore. Access pattern type here refers to a combination of number of updates to file and its depth in file system directory. The x-axis shows the number of Get api calls required for a spatial search during restore. The y-axis shows the fraction of restore requests corresponding to the get requests. This CDP clearly shows that Trie based queueing results in significantly less Get api calls compared to FIFO base queuing for a majority of the restore requests. Table 7.2 shows the breakdown of Trie and FIFO queueing with regards to list, get and useful get api calls (get resulting in actual restore results) and the time taken to search. Numbers shown are average over all files in restore set. Clearly, average number of gets for Trie LCP is significantly lower, about 727.5.

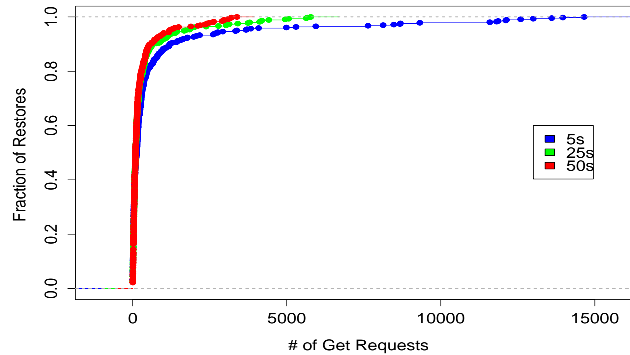


Figure 7.12: Impact of *Flush Interval* on Restore spatial search performance

Where as for FIFO LCP, number of gets is close to 11k. The reducing in total gets leads to much faster search time in case of Trie LCP, about 17.83 seconds compared to FIFO LCP which takes almost 210.5 seconds.

Figure 7.12 shows a impact of *flush interval* on restore spatial search performance for Trie based queuing. For this experiment, one file of each access pattern type of chosen for restore. Access pattern type here refers to a combination of number of updates to file and its depth in file system directory. The x-axis shows the number of Get api calls required for a restore. The y-axis shows the fraction of restore requests corresponding to the get requests. Multiple CDPs each for a different value of flush interval are shown. Clearly, higher the flush interval, better the restore search performance.

Table 7.3 shows the breakdown of Trie and FIFO queueing with regards to list, get and useful get api calls (get resulting in actual restore results) and the time taken to search. Again, the numbers are average across all files in restore fileset. Clearly, average number of gets for Trie LCP reduces with increase in flush interval. Larger *flush intervals* can lead to larger vulnerability windows and/or larger potential data loss. Hence, one needs consider a careful tradeoff between restore performance and vulnerability window while selecting flush intervals.

Table 7.3: Impact of *Flush Interval* on Restore spatial search performance

Api Request Type	*Flush Interval* 5s	*Flush Interval* 25s	*Flush Interval* 50s
List	7.2	7.22	7.27
Get	727.5	427.22	266.64
Useful Get	67.52	63.6	64.9
Time	17.83	13.6	10.58

7.5 Conclusions & Future Work

Temporal and spatial search of data for purposes of restore in CDP based on object storage introduces new unique challenges. In this work we show how uniqueness of these workloads can be mapped to salient characteristics of object storage systems. Specifically, by exploiting data layout organization and name encoding, we show that both temporal and spatial search performance can be significantly optimized. Preliminary results show that Temporal search performance of cCDP with proposed hybrid data layout is within 28% of Btree based temporal data layout and is about 52.7% better than the naive data layout with name encoding with significantly lower edge buffer and write throughput overheads than the Btree based temporal data layout. Further, Spatial search performance of proposed object name encoding and Trie based queueing is about 15X more efficient than naive encoding and 12X faster than naive encoding.

Chapter 8

Conclusions & Future Directions

Optimizing cost of data management is crucial for enterprise IT management. Substantial operational cost efficiencies can be achieved with exploitation of application hints on top of MAID storage systems. Average energy savings achievable with application aided MAID is about 25% compared to traditional MAID arrays. Further, scheduling technique used drastically impacts the amount of energy consumed by the system. On average, our GreenStor storage system using its opportunistic deep prefetcher provides energy savings of an extra 10% or more compared to other scheduling techniques.

Operational cost efficiencies and scalability of traditional data protection systems can be greatly improved using of model based approaches. Our proposed model based scheduling framework takes into account instantaneous server and storage load explicitly and rearranges backup start times to optimize back performance. Preliminary results show that for about 20% of backups, performance improvement is greater than 32%. On an average, improvement in backup completion time is about 13%.

For next generation systems and emerging workloads, CDP enables recoverability to any point in time and caters to a wider variety of data protection requirements. However, the capital costs of CDP using traditional storage architectures is prohibitive, limiting its usage. Our proposed cCDP system design incorporates edge buffering and destage optimizations on top of cloud object storage systems that results in CDP write throughputs comparable to raw block based enterprise storage with very minimal memory requirements ($< 1\%$) with appropriate choice of Coalesce Factor (CF) and concurrency.

Temporal and spatial search of data for purposes of restore in CDP based on object storage introduces new unique challenges. By exploiting data layout organization and name encoding, we show that both temporal and spatial search performance can be significantly optimized. Temporal search performance of cCDP with proposed hybrid data layout is within 28% of Btree based temporal data layout and is about 52.7% better than the naive data layout with name encoding with significantly lower edge buffer and write throughput overheads than the Btree based temporal data layout. Further, Spatial search performance of proposed object name encoding and Trie based queueing is about 15X more efficient than naive encoding and 12X faster than naive encoding.

In this work we do not consider any constraints on disk transitions. It has recently come to our attention that, in MAID systems, disks are packed every closely to minimize the impact of humidity on disk lifetime. Specifically, if disks are not turned on for a long period, the probability of failure on the next spinup increases due to the collection of moisture from humidity. Hence, MAID manufactures pack disks closely to minimize the effects of humidity. In our proposed design, this close packing of disks could lead to overheating in certain regions of the disk array if not properly arbitrated. Approaches by which intelligent arbitration of disk spinups could be used to minimize the problem of overheating is an interesting avenue for future work.

In this work we explored making one aspect of backup policies dynamic - the start time of backups. Other dimensions which we intend to explore as part of future work are client to storage and client to backup server associations. As future work, we also intend to explore storage configurations of backup servers. Deduplication, Tiering and replication on backup servers interact in complex fashion and provide significant room for optimization in a cloud environment.

Dynamic optimization of Coalesce Factor (CF) and concurrency level based on observed workload behavior and performance of underlying object storage cluster is an interesting area of exploration. Impact of edge buffer physical placement on overall CDP design and detailed exploration of deletion and retention requirements are other areas of future work.

References

[1] Epa report on server and data center energy efficiency. `http://www.energystar.gov/index.cfm?c=prod_development.server_efficiency_study`.

[2] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mahmut Kandemir, and Hubertus Franke. Drpm: dynamic speed control for power management in server class disks. In *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*, pages 169–181, New York, NY, USA, 2003. ACM.

[3] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mahmut Kandemir, and Hubertus Franke. Drpm: dynamic speed control for power management in server class disks. In *Proceedings of the 30th international symposium on Computer architecture*, pages 169–181, New York, USA, 2003.

[4] Enrique Carrera Eduardo, Eduardo Pinheiro, and Ricardo Bianchini. Conserving disk energy in network servers. In *In Proceedings of the 17th International Conference on Supercomputing*, pages 86–97, 2003.

[5] Eduardo Pinheiro and Ricardo Bianchini. Energy conservation techniques for disk array-based servers. In *Proceedings of the 18th international conference on Supercomputing*, pages 68–78, New York, USA, 2004.

[6] Qingbo Zhu, Zhifeng Chen, Lin Tan, Yuanyuan Zhou, Kimberly Keeton, and John Wilkes. Hibernator: helping disk arrays sleep through the winter. *SIGOPS Oper. Syst. Rev.*, 39(5):177–190, 2005.

[7] Dennis Colarelli and Dirk Grunwald. Massive arrays of idle disks for storage archives. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–11, Los Alamitos, CA, USA, 2002.

[8] Qingbo Zhu, Francis M. David, Christo F. Devaraj, Zhenmin Li, Yuanyuan Zhou, and Pei Cao. Reducing energy consumption of disk storage using power-aware cache management. In *HPCA '04: Proceedings of the 10th International Symposium on High Performance Computer Architecture*, page 118, Washington, DC, USA, 2004. IEEE Computer Society.

[9] Qingbo Zhu, Asim Shankar, and Yuanyuan Zhou. Pb-lru: a self-tuning power aware storage cache replacement algorithm for conserving disk energy. In *ICS '04: Proceedings of the 18th annual international conference on Supercomputing*, pages 79–88, New York, NY, USA, 2004. ACM.

[10] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. Write off-loading: Practical power management for enterprise storage. *In Processings of 6th USENIX Conference on File and Storage Technologies (FAST)*, pages 253–267, 2008.

[11] Jun Wang, Huijun Zhu, and Dong Li. eraid: Conserving energy in conventional disk-based raid system. *IEEE Transactions on Computers*, 57(3):359–374, 2008.

[12] Charles Weddle, Mathew Oldham, Jin Qian, An-I Andy Wang, Peter Reiher, and Geoff Kuenning. Paraid: a gear-shifting power-aware raid. In *FAST '07: Proceedings of the 5th USENIX conference on File and Storage Technologies*, pages 30–30, Berkeley, CA, USA, 2007. USENIX Association.

[13] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. Design tradeoffs for ssd performance. In *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 57–70, Berkeley, CA, USA, 2008. USENIX Association.

[14] Marco A A Sanvido, Frank R Chu, Anand Kulkarni, and Robert Selinger. Nand flash memory and its role in storage architectures. volume 96, pages 1864–1874. IEEE, IEEE, November 2008.

[15] W. Hutsell. Solid state storage for the enterprise, a snia tutorial. `http://www.snia.org/education/tutorials/storage`, 2007.

[16] J Kim, J Kim, S Noh, S Min, and Y Cho. A space-efficient flash translation layer for compact flash systems. In *IEEE Transactions on Consumer Electronics*, volume 48, 2002.

[17] S Lee, B Moon, C Park, J Kim, and S Kim. A case for flash memory ssd in enterprise database applications. In *SIGMOD*, 2008.

[18] J Sykes. Ssds to boost datacenter performance. `http://download.micron.com/pdf/whitepapers/`.

[19] Dell puts ssd hard drives in latitude d420 and atg d620. `http://www.notebookreview.com/default.asp?newsID=3658`, 2007.

[20] Zeus-iops solid state drives surge to 512gb in standard 3.5 inch form factor; offer unprecedented performance for enterprise computing. `http://www.globenewswire.com/newsroom/news.html?d=117663`, April 2007.

[21] Samsung enterprise solid state drive. `http://www.samsung.com/global/business/semiconductor/products/flash/ssd/2008/down/`, October 2008.

[22] R. H. Patterson, G. A. Gibson, and M. Satyanarayanan. Using transparent informed prefetching to reduce file read latency. In *Proceedings of the IEEE/NASA Mass Storage Systems*, pages 329–342, 1992.

[23] Fay Chang and Garth A. Gibson. Automatic I/O hint generation through speculative execution. In *Proceedings of the 3rd symposium on Operating System Design and Implmentation*, pages 1–14, CA, USA, 1999.

[24] Todd C. Mowry, Angela K. Demke, and Orran Krieger. Automatic compiler-inserted I/O prefetching for out-of-core applications. In *Proceedings of the 1st symposium on Operating System Design and Implementation*, pages 3–17, 1996.

[25] Andreas Weissel, Bjorn Beutel, and Frank Bellosa. Cooperative I/O: a novel I/O semantics for energy-aware applications. *SIGOPS Oper. Syst. Rev.*, 36:117–129, 2002.

[26] Nohhyun Park and David J. Lilja. Characterizing datasets for data deduplication in backup applications. In *IISWC*, pages 1–10. IEEE, 2010.

[27] Fanglu Guo and Petros Efstathopoulos. Building a high-performance deduplication system. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'11, pages 25–25, Berkeley, CA, USA, 2011. USENIX Association.

[28] Deepak R. Bobbarjung, Suresh Jagannathan, and Cezary Dubnicki. Improving duplicate elimination in storage systems. *Trans. Storage*, 2(4):424–448, November 2006.

[29] Erik Kruus, Cristian Ungureanu, and Cezary Dubnicki. Bimodal content defined chunking for backup streams. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, FAST'10, pages 18–18, Berkeley, CA, USA, 2010. USENIX Association.

[30] Sean Quinlan and Sean Dorward. Venti: A new approach to archival data storage. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST '02, Berkeley, CA, USA, 2002. USENIX Association.

[31] Phlip Shilane, Mark Huang, Grant Wallace, and Windsor Hsu. Wan-optimized replication of backup datasets using stream-informed delta compression. *Trans. Storage*, 8(4):13:1–13:26, December 2012.

[32] Wei Dong, Fred Douglis, Kai Li, Hugo Patterson, Sazzala Reddy, and Philip Shilane. Tradeoffs in scalable data routing for deduplication clusters. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, FAST'11, pages 2–2, Berkeley, CA, USA, 2011. USENIX Association.

[33] Mark Lillibridge, Kave Eshghi, Deepavali Bhagwat, Vinay Deolalikar, Greg Trezise, and Peter Camble. Sparse indexing: Large scale, inline deduplication using sampling and locality. In *Proccedings of the 7th Conference on File and Storage Technologies*, FAST '09, pages 111–123, Berkeley, CA, USA, 2009. USENIX Association.

[34] Benjamin Zhu, Kai Li, and Hugo Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, pages 18:1–18:14, Berkeley, CA, USA, 2008. USENIX Association.

[35] Grant Wallace, Fred Douglis, Hangwei Qian, Philip Shilane, Stephen Smaldone, Mark Chamness, and Windsor Hsu. Characteristics of backup workloads in production systems. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST'12, pages 4–4, Berkeley, CA, USA, 2012. USENIX Association.

[36] Nagapramod Mandagere, Pin Zhou, Mark A Smith, and Sandeep Uttamchandani. Demystifying data deduplication. In *Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion*, Companion '08, pages 12–17, New York, NY, USA, 2008. ACM.

[37] Robert Birke, Mathias Bjoerkqvist, Lydia Y. Chen, Evgenia Smirni, and Ton Engbersen. (big)data in a virtualized world: Volume, velocity, and variety in cloud datacenters. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies*, pages 177–189, Santa Clara, CA, 2014. USENIX.

[38] Nitin Agrawal, William J. Bolosky, John R. Douceur, and Jacob R. Lorch. A five-year study of file-system metadata. *Trans. Storage*, 3(3), October 2007.

[39] John R. Douceur and William J. Bolosky. A large-scale study of file-system contents. In *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '99, pages 59–70, New York, NY, USA, 1999. ACM.

[40] Dutch T. Meyer and William J. Bolosky. A study of practical deduplication. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, FAST'11, pages 1–1, Berkeley, CA, USA, 2011. USENIX Association.

[41] Andrew W. Leung, Shankar Pasupathy, Garth Goodson, and Ethan L. Miller. Measurement and analysis of large-scale network file system workloads. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ATC'08, pages 213–226, Berkeley, CA, USA, 2008. USENIX Association.

[42] Priya Sehgal, Vasily Tarasov, and Erez Zadok. Evaluating performance and energy in file system server workloads. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, FAST'10, pages 19–19, Berkeley, CA, USA, 2010. USENIX Association.

[43] Mark Chamness. Capacity forecasting in a backup storage environment. In *Proceedings of the 25th International Conference on Large Installation System Administration*, LISA'11, pages 12–12, Berkeley, CA, USA, 2011. USENIX Association.

[44] Jing Yang, Qiang Cao, Xu Li, Changsheng Xie, and Qing Yang. ST-CDP: Snapshots in TRAP for Continuous Data Protection. *IEEE Transactions on Computers*, 61(6):753–766, 2012.

[45] Xu Li, Changsheng Xie, and Qing Yang. Optimal implementation of Continuous Data Protection (CDP) in linux kernel. In *Networking, Architecture, and Storage, 2008 (NAS'08)*. IEEE, 2008.

[46] Tivoli Continuous Data Protection for files. `http://www-03.ibm.com/software/products/en/tivolicontinuousdataprotectionforfiles`.

[47] Falconstor Continuous Data Protector. `http://falconstor.com`.

[48] EMC Recoverypoint - Application recovery to any point in time. `http://www.emc.com/storage/recoverpoint/recoverpoint.htm`.

[49] Symantec Netbackup Continuous Data Protection. `http://eval.symantec.com/mktginfo/enterprise/tech_briefs/b-techbrief_nbu_snapshots_20719041.en-us.pdf`.

[50] Guy Laden, Paula Ta-Shma, Eitan Yaffe, Michael Factor, and Shachar Fienblit. Architectures for Controller based CDP. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, FAST '07, pages 21–21, Berkeley, CA, USA, 2007. USENIX Association.

[51] Qing Yang, Weijun Xiao, and Jin Ren. TRAP-Array: A disk array architecture providing timely recovery to any point-in-time. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, ISCA '06, pages 289–301, Washington, DC, USA, 2006. IEEE Computer Society.

[52] Charles B. Morrey III and Dirk Grunwald. Peabody: The time travelling disk. In *IEEE Symposium on Mass Storage Systems*, pages 241–253, 2003.

[53] Yonghong Sheng, Dongsheng Wang, Jinyang He, and Dapeng Ju. TH-CDP: an efficient block level continuous data protection system. In *International Conference on Networking, Architecture, and Storage, NAS 2009, 9-11 July 2009, Zhang Jia Jie, Hunan, China*, pages 395–404, 2009.

[54] Maohua Lu, Shibiao Lin, and Tzi-cker Chiueh. Efficient Logging and Replication techniques for Comprehensive Data Protection. In *24th IEEE Conference on Mass Storage Systems and Technologies (MSST 2007), 24-27 September 2007, San Diego, California, USA*, pages 171–184, 2007.

[55] Maohua Lu, Dilip Nijagal Simha, and Tzi-cker Chiueh. Scalable index update for block-level continuous data protection. In *Sixth International Conference on Networking, Architecture, and Storage, NAS 2011, Dalian, China, 28-30 July, 2011*, pages 372–381, 2011.

[56] Craig A. N. Soules, Garth R. Goodson, John D. Strunk, and Gregory R. Ganger. Metadata Efficiency in Versioning File Systems. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, FAST '03, pages 43–58, Berkeley, CA, USA, 2003. USENIX Association.

[57] Huaiming Song, Yanlong Yin, Yong Chen, and Xian-He Sun. A cost-based application-specific data layout scheme for parallel file systems. In *The 20th International Symposium on High Performance Distributed Computing (HPDC'11)*, pages 37–48, 2011.

[58] Zhenhuan Gong, David A. Boyuka II, Xiaocheng Zou, Qing Liu, Norbert Podhorszki, Scott Klasky, Xiaosong Ma, and Nagiza F. Samatova. Parlo: Parallel runtime layout optimization for scientific data explorations with heterogeneous access

patterns. *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 0:343–351, 2013.

[59] Shuibing He, Yan Liu, and Xian-He Sun. PSA: A Performance and Space-aware data layout scheme for hybrid parallel file systems. In *Proceedings of the 2014 International Workshop on Data Intensive Scalable Computing Systems*, DISCS '14, pages 41–48, Piscataway, NJ, USA, 2014. IEEE Press.

[60] Hsuan-Te Chiu, Jerry Chou, Venkat Vishwanath, Suren Byna, and Kesheng Wu. Simplifying Index File Structure to improve i/o performance of Parallel Indexing. In *The 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS 2014)*, 2014.

[61] Wenting Zheng, Stephen Tu, Eddie Kohler, and Barbara Liskov. Fast databases with fast durability and recovery through multicore parallelism. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 465–477, Berkeley, CA, USA, 2014. USENIX Association.

[62] John D. Strunk, Garth R. Goodson, Michael L. Scheinholtz, Craig A. N. Soules, and Gregory R. Ganger. Self-securing storage: Protecting data in compromised system. In *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation - Volume 4*, OSDI'00, pages 12–12, Berkeley, CA, USA, 2000. USENIX Association.

[63] Paula Ta-Shma, Guy Laden, Muli Ben-Yehuda, and Michael Factor. Virtual Machine Time Travel using Continuous Data protection and checkpointing. *SIGOPS Oper. Syst. Rev.*, 42(1):127–134, January 2008.

[64] Openstack Swift object storage. `http://docs.openstack.org/developer/swift/`.

[65] Amazon S3 object storage. `http://aws.amazon.com/s3/`.

[66] Google object storage. `https://cloud.google.com/storage/docs/overview`.

[67] American Power Corporation(APC). Determining total cost of ownership for data center and network room infrastructure. www.apcmedia.com/salestools/CMRP-5T9PQG_R3_EN.pdf, 2004.

[68] M Hopkins. The onsite energy generation option, the data center j. http://datacenterjournal.com/News/Article.asp?article_id=66, 2004.

[69] K. Rajamani and C. Lefurgy. On evaluating request-distribution schemes for saving energy in server clusters. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, pages 111–122, CA, USA, 2003.

[70] David Rochberg and Garth Gibson. Prefetching over a network: early experience with ctip. *SIGMETRICS Perform. Eval. Rev.*, 25(3):29–36, 1997.

[71] Andrew Tomkins, R. Hugo Patterson, and Garth Gibson. Informed multi-process prefetching and caching. In *Proceedings of the ACM SIGMETRICS*, pages 100–114, NY, USA, 1997.

[72] Pei Cao, Edward W. Felten, Anna R. Karlin, and Kai Li. A study of integrated prefetching and caching strategies. *SIGMETRICS Perform. Eval. Rev.*, 23(1):188–197, 1995.

[73] Nagapramod Mandagere, Jim Diehl, and David Du. Greenstor: Application-aided energy-efficient storage. In *MSST '07: Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies*, pages 16–29, Washington, DC, USA, 2007. IEEE Computer Society.

[74] Nagapramod Mandagere, Jim Diehl, and David Du. Greenstor: Application-aided energy-efficient storage. Technical report, Digital Technology Center, Univ of Minnesota, 2007.

[75] Ibm business continuity and resiliency services. http://www.ibm.com/bcrs.

[76] Support vector machines for classification & regression. http://www.kernel-machines.org/.

[77] David Du Nagapramod Mandagere, Ramani Routray. Towards a framework for adaptive backup management - work in progress. In *FAST '14: Proceedings of the*

12th USENIX Conference on File and Storage Technologies, page 7, Santa Clara, CA, USA, 2014. USENIX Association.

[78] Computing as a service over the internet. http://www.ibm.com/cloud-computing/us/en/what-is-cloud-computing.html.

[79] Predicts 2015: Cloud Computing goes beyond IT into Digital Business. https://www.gartner.com/doc/2922018.

[80] What is changing in the realm of big data. http://www.ibm.com/big-data/us/en/.

[81] Gartner: Driving value from Big Data. http://www.gartner.com/technology/research/big-data/.

[82] Yonghong Sheng, Dongsheng Wang, Jin-Yang He, and DaPeng Ju. TH-CDP: An efficient block level Continuous Data Protection system. In *Networking, Architecture, and Storage, 2009 (NAS 2009)*, pages 395–404. IEEE, 2009.

[83] IBM Softlayer object storage. http://www.softlayer.com/object-storage.

[84] Rackspace cloud files. http://www.rackspace.com/cloud/files.

[85] vCloud Air - Google Object storage integration. http://blogs.gartner.com/kyle-hilgendorf/2015/01/29/vcloud-air-google-cloud-platform-an-ideal-marriage-for-now/.

[86] Filebench - filesystem and storage benchmark. http://filebench.sourceforge.net.

[87] FUSE - Filesystem in User Space. http://fuse.sourceforge.net.

[88] Redis - key value cache and store. http://redis.io.

[89] Golang - the Go programming language. https://golang.org.

[90] Nagapramod Mandagere, Ramani Routray, Yang Song, and David Du. Cloud object storage based continuous data protection (ccdp). In *Networking, Architecture and Storage (NAS), 2015 IEEE International Conference on*, pages 244–254. IEEE, 2015.

[91] Trie data structure. https://en.wikipedia.org/wiki/Trie.

Appendix A

Acronyms

Table A.1: Acronyms

Acronym	Meaning
MAID	Massive Array of Idle Disks
SDS	Software Defined Storage
CDP	Continuous Data Protection
RPM	Rotations Per Minute
SSD	Solid State Disks
SCSI	Small Computer Systems Interface
iSCSI	Internet Small Computer Systems Interface
SATA	Serial Attach ATA
SAS	Serial Attach SCSI
DRPM	Dynamic Rotations Per Minute
LRU	Least Recently Used
PiT	Point in Time
HVAC	Heating, Ventilation & Air Conditioning
EPA	Environment Protection Agency
SAN	Storage Area Network
HPC	High Performance Computing

Continued on next page

Table A.1 – continued from previous page

Acronym	Meaning
LBA	Logical Block Address
RAID	Redundant Array of Independent Disks
FTL	Flash Translation Layer
DRAM	Dynamic Random Access Memory
HTTP	Hyper Text Transfer Protocol
REST	Representational State Transfer
TRAP	Timely Recovery to Any Point
LUN	Logical Unit Number
FCFS	First Come First Serve
FIFO	First In First Out
LCP	Longest Common Prefix
SLA	Service Level Aggrement
RPO	Recovery Point Objective
RTO	Recovery Time Objective
SVR	Support Vector Regression
MLR	Multi Linear Regression
CDN	Content Delivery Network
FUSE	File system in User Space
NVRAM	Non Volatile Random Access Memory