Scaling data capacity and throughput in encrypted deduplication with segment chunks and index locality

Scaling Data Capacity and Throughput in Encrypted Deduplication with Segment Chunks and Index Locality

Jaybe Ammons

2024

A thesis submitted to the University of London
for the degree of Doctor of Philosophy

Birkbeck, University of London
School of Computing and Mathematical Sciences

Declaration

This thesis is the result of my work, except where explicitly acknowledged in the text.

Abstract

Encrypted deduplication backup systems play a critical role in modern data management by enhancing storage efficiency while ensuring data security. However, they face challenges such as excessive metadata storage in long-term backups and deduplication indexes that exceed available server memory. Heavy backup workloads often experience reduced throughput due to resource contention when concurrently deduplicating multiple client backup streams.

This study introduces the SCAIL suite of algorithms – SCAIL, R-SCAIL, and their multiprocessor adaptations, P-SCAIL and PR-SCAIL – which address these issues. These algorithms significantly reduce metadata storage and memory usage while mitigating resource contention. These optimisations enable a substantial scale-up of both data volume and concurrent client capacities, extending well beyond the limitations of conventional encrypted deduplication methods. Moreover, the SCAIL algorithms uniquely combine the data throughput advantages of coarse-grained *segment*-based deduplication with the high data compression of fine-grained *chunk*-based deduplication.

The SCAIL suite adapts Metadedup's approach to metadata deduplication for client-side deduplication by employing a memory-based index. This index utilises fingerprints generated from the metadata of data segments to identify duplicate content efficiently. This strategy enables the rapid elimination of duplicate segments, significantly streamlining the deduplication process while reducing metadata uploads and the overall storage footprint. With this coarse level of deduplication, SCAIL may sometimes reupload previously saved chunks. To mitigate this, R-SCAIL introduces resemblance-based, chunk-level client-side deduplication, effectively reducing redundant uploads. This refinement trades some of SCAIL's speed for reduced upload volume, resulting in R-SCAIL operating at a slower throughput.

For server-side deduplication, the SCAIL family adapts Sorted Deduplication's index locality technique to perform exact, cross-client chunk-level deduplication in a

very efficient single sequential pass through the disk-based index.

By harnessing multiprocessor server architectures, P-SCAIL and PR-SCAIL introduce data and task parallelism, significantly boosting throughput for deduplication processes both on client-side and server-side deduplication.

Our evaluation with two widely used public backup datasets shows that the SCAIL suite significantly reduces memory and storage requirements, thereby enhancing server capacity to manage larger data volumes and support more concurrent clients. P-SCAIL reached hundreds of GiB/second in client-side deduplication, and PR-SCAIL reached a range of tens of GiB/second. Both systems are compatible with the throughput transfer rates of modern hard-disk drives during server-side deduplication. The resulting high throughput, low memory, and storage requirements of the SCAIL family significantly advance the field of encrypted deduplication.

Acknowledgements

The academic team at Birkbeck, University of London have provided input throughout this work.

I would like to extend my profound gratitude to my supervisors, Trevor Fenner and David Weston. Trevor, after guiding me through my Master's thesis on secure deduplication, not only encouraged me to delve deeper with a PhD but also introduced me to David, an exemplary co-supervisor. Together, Trevor and David have been instrumental in shaping my academic journey, offering invaluable insights on my conference and journal paper submissions and providing unwavering support throughout my research for this thesis.

Engaging in discussions with fellow students improved my work. I'd like to extend special gratitude to Bernard Fromson for our weekly discussions during the extended days of the COVID lockdowns. These conversations allowed me to articulate and refine my ideas, and I'm grateful for his generosity in sharing tools and techniques with me.

My wife, Shelly, has been supportive throughout and provided encouragement and comfort.

Contents

List of Figures

List of Tables

Acronyms

P-SCAIL **P**arallel SCAIL. 3, 4, 16, 17, 20, 32, 33, 35, 112, 113, 115, 118, 120–123, 125–128, 130–134, 136, 139, 140, 145, 146, 148–151, 153, 154, 156–159, 161–164, 166–182, 188–192, 194–198, 202, 205, 209, 210, 212–214, 216

PR-SCAIL **P**arallel **R**esemblance SCAIL. 3, 4, 16, 17, 20, 21, 33–35, 130, 134, 136, 139–141, 144–146, 148–155, 157–159, 161–164, 166–174, 176, 177, 180–182, 186, 188, 189, 191, 193, 195–198, 205, 208–214, 216

R-SCAIL **R**esemblance SCAIL. 3, 18, 19, 33, 183–188

CDC Content-Defined Chunking. 26, 37, 77, 86, 107, 131, 132

CFP Chunk Fingerprint. 14, 15, 26, 30, 36, 38, 39, 42, 43, 49, 52, 53, 58, 63–66, 68, 77, 78, 80, 83, 87, 94–96, 98, 102, 108, 113–115, 118, 120, 121, 125, 126, 134, 136, 138, 139, 144, 147, 148, 150, 159, 161–164, 166, 167, 171, 184, 185, 195, 209, 214, 215

CID Container ID. 64, 65, 83, 87, 90, 114, 115, 118, 121, 136, 138, 144, 161, 162

DBA Dynamic Bloom filter Array. 68

DER Duplicate Elimination Ratio. 49

LSM tree Log-Structured Merge-Tree. 87

MFP Metachunk Fingerprint. 21, 30, 52, 53, 58, 60, 74, 77, 78, 80, 83, 86, 94, 96–98, 109, 113, 120, 121, 125, 126, 131, 134, 136, 138, 140, 144, 147, 150, 153, 161–164, 166, 167, 184, 206–208, 210, 211, 214

MLE Message-Locked Encryption. 31, 40, 42, 43, 77, 94, 95, 97, 99, 102, 118, 174, 207, 213

RFP Representative Fingerprint. 21, 134, 136, 138, 141, 153, 162, 166, 211, 214

RSD Redundant Segment Data. 16, 33, 34, 91–93, 109, 126, 130–134, 146, 148, 151, 152, 154, 157–159, 170–173, 175–181, 184, 191, 199, 201–203, 212, 213

SCAIL **S**egment **C**hunks **A**nd **I**ndex **L**ocality. 3, 4, 15–19, 31–35, 43, 44, 47, 58, 71, 73–75, 80, 86–95, 98–106, 108–113, 115, 118–123, 125, 126, 128, 130, 131, 141, 144, 145, 157, 158, 162–169, 173, 174, 178, 180, 182–188, 190, 195, 197, 198, 200, 202–206, 209–213, 216

SCI Sorted Chunk Indexing. 29, 31, 32, 61, 63, 66, 67, 73, 80, 87, 93, 94, 96, 103, 104, 107–109, 112–115, 118, 119, 124–126, 128, 156, 159, 161, 162, 207–209, 212, 213, 215

Chapter 1

Introduction

1.1 Problem settings

Encrypted data deduplication techniques significantly reduce server storage requirements for archival and backup workloads while preserving data privacy [8]. These techniques distinguish between *logical data* — the original client data before deduplication — and *physical data*, which consists of deduplicated data chunks stored on the server. Figure 1.1 shows the growth comparison between these two types of data. The chart on the left shows the growth of logical volume over 115 backups of the FSL Dataset to 56.24 TiB. The chart on the right uses a different scale, and shows how physical (unique, deduplicated) data grows to 431.91 GiB through the same 115 backups. The dashed line at the bottom of the left-hand chart is the volume of physical data projected onto the chart of the logical data volume. For a detailed description of the FSL Dataset, see Subsection 3.12.1.

The significant disparity between logical and physical data size underscores the effectiveness of deduplication techniques, quantified by the Duplicate Elimination Ratio (DER). The DER, calculated as the ratio of logical data to physical data size, in this case, is 133.33, which means that for every unit of physical storage used, 133.33 units of logical data are represented. This high DER value emphasises the potential storage savings, particularly in environments with high data redundancy. Understanding and maximising DER is crucial for optimising deduplication systems, as it directly impacts

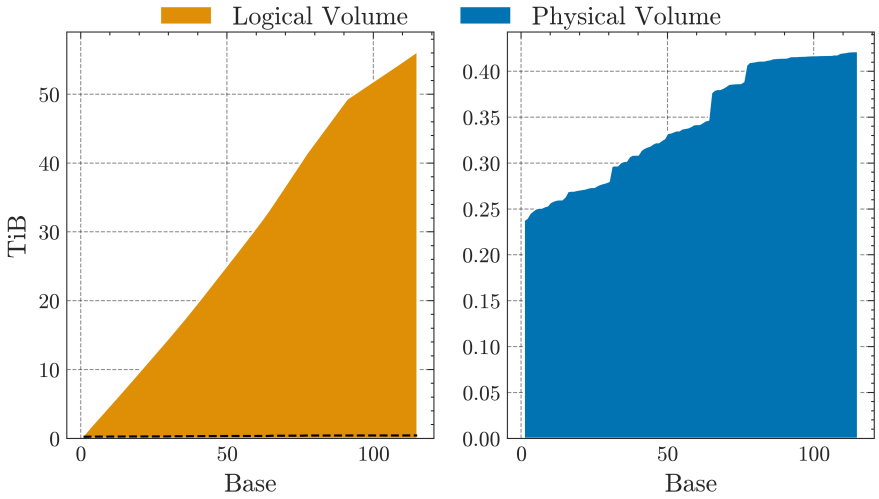


Figure 1.1: Data deduplication significantly reduces server storage requirements. The left area chart illustrates the growth of logical (pre-deduplication) data, while the right chart shows the growth of physical (deduplicated) data for the FSL dataset. Note that the two charts use different scales. The dashed line on the left projects physical data onto the logical scale for comparison.

the storage cost and efficiency.

To facilitate a backup, clients convert file data into variable-sized chunks using one of many *Content-Defined Chunking (CDC)* techniques (see Subsection 2.1.1), encrypt each chunk, and generate a cryptographic hash of the encrypted chunk to create a *chunk fingerprint* (CFP), serving as a compact identifier for the chunk. Lists of CFPs and encrypted keys (further encrypted with a user-specific key), known as *File and Key Recipes* for files, are compiled by the client and stored on the server for retrieval.

Two primary strategies are employed in the data deduplication domain: *client-side* and *server-side* deduplication. Both strategies involve the lookup of Chunk Fingerprints (CFPs) on the server to detect duplicate data. Despite its name, client-side deduplication is effected on the server through coordination with the client. The client starts the process by transmitting CFPs from a file recipe to the server. The server, in turn, identifies chunks that have been previously stored and returns a list of fingerprints corresponding to "Missing" chunks—those not already stored. Consequently, only these missing chunks are uploaded by the client, significantly reducing upload volumes.

Conversely, server-side deduplication requires transmitting all chunk data from the client to the server. Upon receiving this data, the server performs duplicate detection by looking up the fingerprints of the incoming chunks in an index of existing chunks, adding the fingerprints of previously unseen chunks to the index. Through this process, the server ensures that only one copy of each unique chunk is retained, effectively eliminating duplicates post-upload.

For restoration operations under both deduplication strategies, clients request the encrypted chunks from the server using fingerprints derived from the file recipes. The clients then decrypt these chunks using the keys provided in the key recipes, facilitating the reconstruction of the original files.

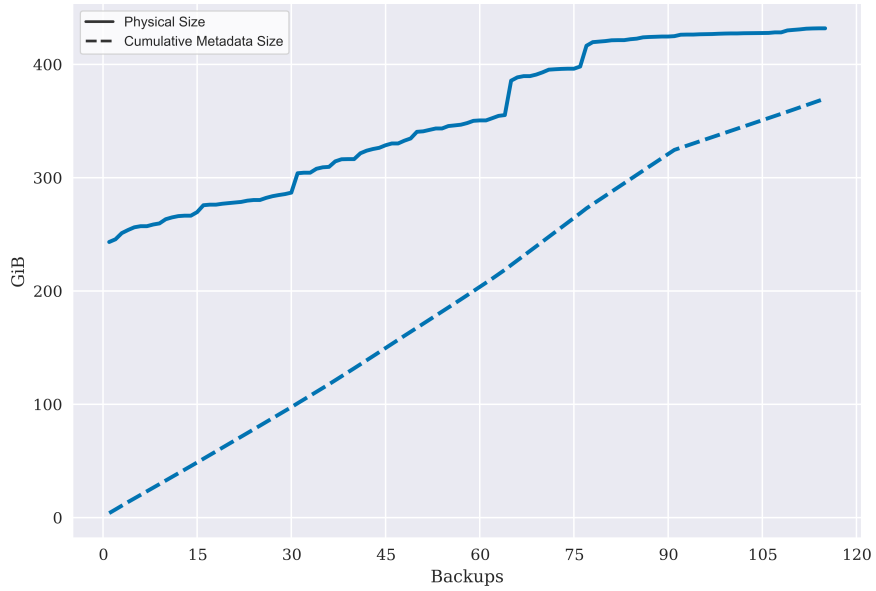


Figure 1.2: The line chart shows metadata storage (dashed line) growing to nearly match the size of data storage (solid line) for the FSL Dataset (see Subsection 3.12.1 for dataset details).

Long-duration backup workloads lead to a considerable accumulation of File and Key Recipe storage. As illustrated in Figure 1.2, the storage required for *metadata*, predominantly consisting of File and Key Recipes, starts out modestly at approximately 4 GiB but scales with the logical data. In contrast, the physical data starts at 243.32 GiB and demonstrates gradual growth. This difference highlights a critical aspect of

deduplication systems. Even if the growth in storage required is slow, the associated metadata continues to grow rapidly, underscoring the need for efficient metadata management alongside data deduplication strategies.

Metadedup, introduced by Li et al. [44] addressed this issue by gathering sequences of encrypted chunks into segments. The metadata from the segment chunks are combined to produce a chunk of metadata, which we term a *metachunk*. Taking the hash of the encrypted metachunk produces a *metachunk fingerprint* (MFP) that is used to identify it and is added to the chunk fingerprint index. The encrypted metachunk is included in the backup data and thus is deduplicated in the same way as the chunk data. Metadedup used a *two-phase* deduplication process that performed cross-user deduplication only for server-side deduplication, not client-side deduplication.

A further challenge arises as the accumulated volume of deduplicated data stored on the server increases. Eventually, it will become infeasible to hold the fingerprint index in DRAM (which we refer to as *memory* throughout this document). Resorting to a disk-based index would generate random disk seeks, increasing response time to clients in answer to their duplicate lookup queries. Kaiser et al., in their paper on *Sorted Deduplication* [36] noted that algorithm designers have used at least two approaches to address this issue (the following quotation is italicised, the citations have been changed to those in our bibliography):

- *They reduce the index size so that it fits in main memory and trade this for a reduced duplicate detection rate.*
- *They exploit chunk locality, i.e. the tendency for chunks in backup data streams to reoccur together [49] and prefetch chunk fingerprints block-wise from an internal data structure that catches this locality. For a single backup stream, prefetching can generate a near-sequential disk access [37].*

They noted that each stream has its distinct locality, generating contention for disk access and cache space and that as more client streams are processed concurrently, the

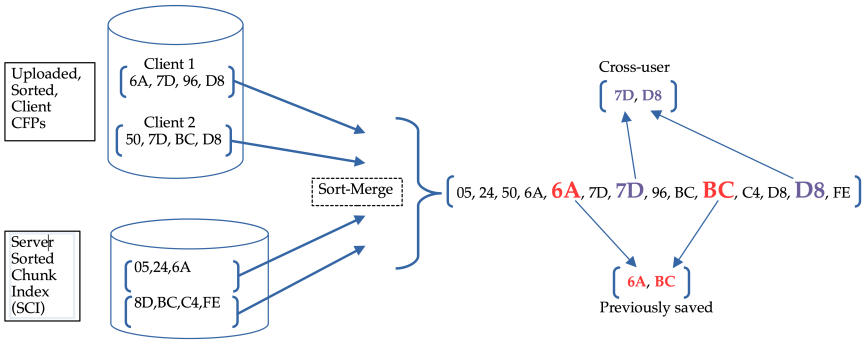


Figure 1.3: Sorted Deduplication's [36] Sorted Chunk Indexing (SCI) Example: Multiple clients' sorted-CFP lists are merged with CFPs of previously saved chunks in SCI bins. Any repeated CFPs found in the merge are either in multiple client uploads (cross-client duplicates 7D and D8) or are previously saved duplicates (6A and BC).

performance of the chunk locality schemes will degrade. They introduced the Sorted Chunk Indexing (SCI) design shown in Figure 1.3, which can process all client backup streams in a single pass with minimal disk I/O and low memory requirements. The authors note, however, that the system was designed for performance rather than security, so it would be susceptible to various data privacy attacks in an unsecured environment.

1.2 Objectives

We aspire to establish the feasibility of a single-server, exact, encrypted deduplication system with the capacity and throughput to handle petabyte-scale datasets, capable of supporting hundreds of concurrent client backups while reducing both upload volume and storage requirements.

Specifically, our objectives are to:

1. **Reduce Server Storage**: Reduce server storage requirements by deduplicating metadata as well as performing exact, chunk-level deduplication at scale.

2. **Reduce Memory Requirements**: Reduce server memory requirements associated with processing client-side and server-side deduplication, aiming to miti-

gate disk bottlenecks and support petabyte scalability on a single server.

3. **Fast Client-side Deduplication**: Enable high-speed throughput for client-side deduplication, demonstrating the feasibility of processing client backups in the petabyte-scale of unique (deduplicated) data.

4. **Ensure Data Privacy**: Provide strong data privacy guarantees against brute force and other attacks, and ensure resistance to side-channel attacks.

5. **Reduce Resource Contention**: Reduce, and if possible, eliminate resource contention during client-side and server-side deduplication.

6. **Increase Throughput with Parallelism**: Increase deduplication throughput by leveraging parallelism on multiprocessor systems.

7. **Reduce Upload Volume**: Achieve reduced upload volumes by incorporating metadata deduplication and resemblance techniques.

1.3 Contributions

To achieve these objectives, we made the following contributions:

1. **Metachunk Fingerprint (MFP) Index**: We introduce the novel concept of a client-side deduplication index that uses *only the hash digest* (the Metachunk Fingerprint (MFP)) of segments (groups of chunks) rather than individual chunk fingerprints. To the best of our knowledge, no other deduplication scheme employs this approach (the Metadedup index contains both CFPs and MFPs.) This reduces the memory required for client-side deduplication by up to 250 times, helping achieve Objective 2.

   This substantially reduced index size allows us to process large datasets within memory, achieving single-processor, single-server client-side deduplication through-

put of up to 100 GiB/second. This manifests as rapid response times for client deduplication lookup queries, assisting us in achieving Objective 3.

The elimination of duplicate metachunks before storing them on the server reduces their storage volume by up to 97%, which helps reduce overall storage on the server by up to 44%, achieving Objective 1.

Since the SCAIL family encrypts metachunks with MLE, this contributes to achieving Objective 4.

2. **Secure Sorted Deduplication**: We adapt Sorted Deduplication's multi-client, batch-oriented Sorted Chunk Indexing (SCI) concept into our chunk-level, server-side deduplication to provide very low memory and disk I/O requirements for chunk-level deduplication at scale, fulfilling Objectives 3 and 5. We ensure that our design allocates chunks to data containers in recipe order to reduce fragmentation, which assists in lowering file restore times.

   Sorted Deduplication was designed with throughput performance as a primary objective without considering data security, so we eliminated the attack-vulnerable technique of client and server-shared construction of file recipes in our design, assisting in achieving Objective 4.

3. **Hybrid Two-Phase Deduplication**: Our hybrid approach combines different deduplication techniques within the two-phase structure to enhance flexibility and performance. It avoids cross-user deduplication in client-side deduplication, which supports Objective 4 by reducing side-channel attack risks.

   In addition, since SCAIL aligns segment boundaries on chunk boundaries, it enables us to combine the capacity and throughput advantages of client-side segment-level deduplication with the storage reduction of server-side chunk-level deduplication. As an example, our two-phase deduplication framework can use 2 MiB segments for initial coarse-grained client-side deduplication, fol-

lowed by cross-client server-side deduplication using fine-grained 8 KiB chunks. Our approach significantly reduces server metadata storage, memory usage, processing time and, typically, the volume of client uploads, contributing to Objectives 1, 2, and 7.

4. **SCAIL Prototype Implementation and Measurements**: We implement prototypes of competitive schemes and our design improvements, and gather measurements on two *trace-based*, publicly available, real-world backup datasets. We empirically compared the efficiency and performance metrics of existing designs against our own, specifically regarding memory usage, server storage volume, upload volume, and disk I/Os.

   Our results demonstrate the feasibility of eliminating all disk I/O for deduplication index access at petabyte scales for client-side deduplication. We achieve reduced upload volumes for server-side deduplication and a very low number of disk I/Os, (at most 350 disk I/Os per backup generation in our evaluations), enabling high-speed chunk-level deduplication and decreased storage volumes from deduplicating metadata.

To ensure that servers have sufficient throughput to take advantage of the increased capacities enabled by SCAIL, we made the following contributions in P-SCAIL:

5. **Improved Cache Management**: We designed an optimised cache management strategy, which reduced SCI cache size used for server-side deduplication processing on the MS dataset from 1.875 GiB to 128 MiB–a 93% reduction–compared to SCAIL, helping achieve Objective 2.

6. **Data and Task Parallelism**: We utilised data and task parallelism to take advantage of multiprocessor servers, achieving speedups of up to 434 GiB/second with 16 processors in client-side and 91 GiB/second in server-side deduplication throughput, achieving Objective 6.

7. **P-SCAIL Prototype Implementation and Measurements**: We implement a multiprocessor capable prototype system for P-SCAIL, and conduct extensive evaluations on our two real-world datasets, focusing on the effect of the number of available processors and varying the segment size on throughput for both client-side deduplication and server-side deduplication. Our results found the parallelisation of the algorithm significantly increases throughput. We observed up to a 60% increase in throughput for each doubling of available processors from 1 up to 16. We also found that increasing segment size can significantly increase client-side deduplication throughput (95 GiB/second at 512 KiB segments to 434 GiB/second at 16 MiB segments).

To address the redundant uploads generated by the SCAIL and P-SCAIL algorithms, we introduce the **R**esemblance SCAIL (R-SCAIL) and **P**arallel **R**esemblance SCAIL (PR-SCAIL) techniques to minimise these excess uploads.

8. **Reduce Redundant Segment Data Upload**: R-SCAIL introduces a novel combination of exact segment and near-exact chunk client-side deduplication to address the Redundant Segment Data (RSD) upload volume inherent in the SCAIL design. By applying resemblance detection techniques at the segment level, R-SCAIL efficiently identifies similar segments and performs client-side chunk-level deduplication with a low-memory footprint. Our results demonstrate significant reductions in upload volume, eliminating up to 97% of RSD. This achieves Objective 7.

9. **Segment Resemblance Task Parallelism**: We introduce parallelism to the resemblance detection process to increase the throughput of client-side R-SCAIL deduplication in PR-SCAIL. This throughput increase from 11 GiB/second to 68 GiB/second helps us achieve Objective 6.

10. **PR-SCAIL Prototype Implementation and Measurements**: We implement a prototype system that can make use of segment resemblance in PR-SCAIL, vary seg-

ment sizes and measure single and multiple processor throughput, memory use and index size, and the volume of RSD, on our two real-world datasets. We found that, while the number of available processors has a big impact on client-side PR-SCAIL deduplication, segment size increase produced almost no effect.

1.4 Publications

The following publication by the author related to this thesis is:

- J. Ammons, T. Fenner, and D. Weston, "SCAIL: Encrypted Deduplication With Segment Chunks and Index Locality," in 2022 IEEE International Conference on Networking, Architecture and Storage (NAS), IEEE, 2022, pp. 1–9.

1.5 Thesis Structure and Methodology

This thesis is organised into seven main chapters, each focusing on a different aspect of Encrypted Deduplication. The structure is designed to provide a logical progression from the introduction of the problem to the detailed analysis of the proposed solutions and their evaluation.

In this chapter we introduced the motivation for this research, the challenges faced in encrypted deduplication, and the contributions made by this thesis.

Chapter 2 lays the foundational concepts necessary for understanding the topics discussed in this thesis, including data deduplication techniques, encryption in deduplication, and the associated challenges. It then examines related works, providing context for the research and highlighting the gaps that this thesis aims to fill.

Chapter 3 presents SCAIL, the new algorithm proposed in this thesis, and describes it in detail.

Chapter 4 parallelises SCAIL, specifically targeting enhancements both in system throughput and response times. This includes the adoption of an optimised caching

strategy and the implementation of task and data parallelism within the SCAIL framework, aimed at streamlining processing efficiency.

Chapter 5 introduces PR-SCAIL, an enhancement to P-SCAIL's client-side deduplication that utilises resemblance techniques to reduce upload volume.

Chapter 6 offers a detailed comparative analysis of the proposed solutions against existing methods using extensive datasets and workloads. This chapter also synthesises the insights gained from the analysis and offers recommendations for various scenarios.

Chapter 7 concludes the thesis by summarising the innovations and advancements introduced, acknowledging the limitations of the current work, and suggesting directions for future research.

The methodology adopted in this thesis is primarily experimental. We begin by reviewing the literature and existing technologies to understand the limitations of current deduplication systems and identify opportunities for enhancements or synergistic combinations of approaches. Subsequently, we utilise trace-driven simulations, employing real-world operational data traces to mimic authentic system usage and assess the performance of our algorithms. This approach ensures our findings are relevant and directly applicable to real-world scenarios. The research is iterative, with each experimentation stage informing subsequent refinements to our algorithms and their evaluation.

Chapter 2

Related Work

Data deduplication is a critical technology in storage systems, particularly in the context of backups and archival storage. By eliminating redundant data, deduplication techniques significantly reduce storage requirements and improve efficiency. However, the integration of encryption for data privacy introduces new challenges, as traditional deduplication methods are rendered ineffective when data is encrypted. This chapter provides a comprehensive overview of data deduplication, discusses the challenges posed by encrypted data, and examines representative designs and current approaches that have shaped the field. The discussion culminates in identifying limitations in existing methods, setting the stage for the solutions proposed in this thesis.

2.1 Data Deduplication

Historically, backups transitioned from tape drives to hard disk drives (HDDs), enabling random access and the identification of duplicate data segments [58, 72, 81]. Deduplication can be extended beyond just comparing files, it can also be used to identify identical portions within files. To detect sub-file duplicates, files are divided into chunks, which are compared using cryptographic hashes known as Chunk Fingerprint (CFP)s. These CFPs serve both as identifiers for duplicate detection and as references for reconstructing the original data during restoration.

2.1.1 Chunking and Fingerprinting

Early deduplication systems, such as Venti [68] and Farsite [23], utilised fixed-size blocks for chunking. However, fixed-size chunking was highly susceptible to data shifts caused by insertions or deletions, which resulted in reduced deduplication effectiveness. To address this limitation, *Content-Defined Chunking* CDC was introduced, defining chunk boundaries based on the intrinsic properties of the data, thereby providing *shift-resistance* [69, 55, 15].

Most Content-Defined Chunking (CDC) techniques are based on Rabin fingerprinting [69], a method that represents data as a polynomial modulo an irreducible polynomial. This technique is chosen for its computational efficiency and its ability to work well in sliding window mechanisms, where the fingerprint of data can be quickly updated as the window moves. In practice, Rabin fingerprinting scans data byte-by-byte using a fixed-size, overlapping window. A "fingerprint," or numerical value, is calculated for each window, which serves as a unique identifier for that section of data.

The Basic Sliding Window technique, popularised by Muthitacharoen et al. [61], computes Rabin fingerprints as a small sliding window (e.g., 64 bytes) moves through the file. Chunk boundaries, or "anchors," are set when the fingerprint matches a specific pattern, such as when a certain number of its lowest bits are zero. For instance, to create chunks with an average size of 8KiB, an anchor is designated when the lowest 13 bits of the fingerprint are zero, since 2^{13} bytes equal 8KiB. This technique allows for chunk boundaries to be detected based on the data itself, making it resistant to shifts caused by insertions or deletions. As a result, even when data positions change, identical portions of data are still detected and deduplicated, effectively reducing data redundancy.

Many different improvements to the Basic Sliding Window technique have been proposed. Some approaches limit the variability of chunk sizes while preserving shift resistance (e.g. [25, 43, 72, 80, 3, 11]). Others use variable chunk sizes in regions of

change, then combine them [12, 43]. Another approach is to re-chunk changed (i.e. non-duplicate) chunks (e.g. [43, 71, 53, 108]). Other approaches focus on improving chunking speed (e.g. [105, 95, 2, 100]). Some parallelised approaches have also sped up chunking but have sacrificed some deduplication efficiency, as in P-Dedupe [91], SS-CDC [64], and QuickCDC [96].

Once the chunk has been identified, a hash can be taken to identify it. This hash should have a high resistance to collision. 48-bit MD5 hashes were initially used, but today SHA-1 hashes [61] are typical. Many approaches have explored techniques to speed up hash computation. Some have taken advantage of multiprocessor systems, creating pipelines (e.g. [51, 94, 31, 93]). Others have integrated the deduplication process into GPGPU methods and accelerated fingerprint generation (e.g. [41, 30, 10, 39]).

2.1.2 Deduplication Location

Deduplication can significantly reduce the volume of data transmitted from clients to backup servers [61, 74, 56]. There are two primary approaches to deduplication: client-side and server-side.

In client-side deduplication, as illustrated in Figure 2.1, clients generate chunk fingerprints (CFPs) and send them to the server. The server identifies which chunks are already stored and returns a list of missing chunks. Consequently, clients need only upload new or modified data, thereby conserving bandwidth and reducing network load. This approach is desired in environments where network bandwidth is limited or costly. However, it needs additional processing on the client side to compute fingerprints and manage chunk metadata.

Alternatively, in server-side deduplication, clients send all data to the server without any prior deduplication processing. The server then performs the deduplication process, identifying and eliminating redundant data blocks. This method simplifies

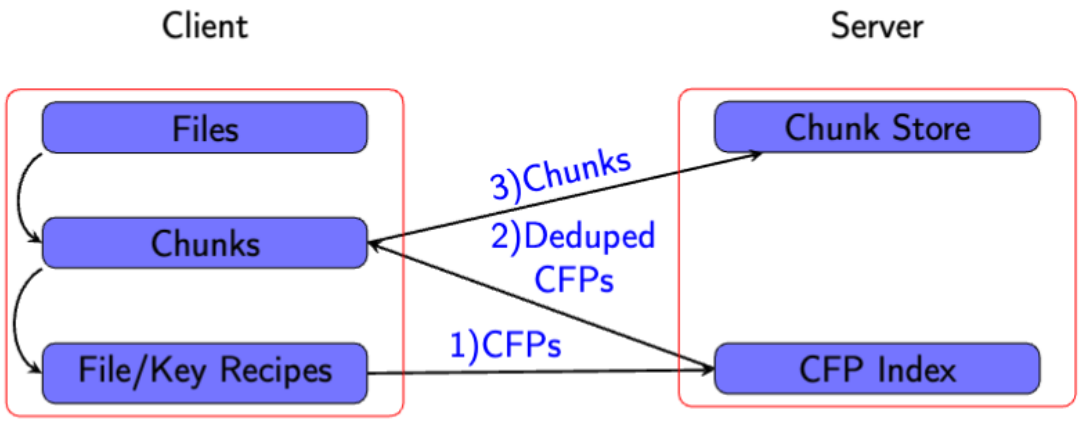


Figure 2.1: Client-side Deduplication in 3 stages. Chunk Fingerprint (CFP)s are sent to the server, which returns the list of chunks not yet stored. The client uploads these chunks and the server stores them.

the client design by offloading the computational overhead of deduplication to the server. Server-side deduplication is beneficial in scenarios where client resources are constrained or when network bandwidth is ample and inexpensive. It centralises the deduplication effort, allowing for more sophisticated and resource-intensive algorithms to be employed on the server. However, this approach can lead to increased network traffic, as all data—including duplicates—must be transmitted to the server.

2.2 Encrypted Deduplication

Encrypting data before transmission and storage is essential for preserving privacy, especially over untrusted networks. However, encryption complicates deduplication because identical plaintexts encrypted with different keys produce different ciphertexts, effectively eliminating the possibility of duplicate detection. Convergent Encryption [23] was developed to enable different clients to encrypt the same chunks with the same key without requiring a key server. It has been studied in many encrypted deduplication designs (e.g., [1, 4, 19, 23, 67, 75, 83, 76]).

The encryption key is derived directly from the chunk itself, typically through a hashing process. Utilising identical keys for encryption yields identical ciphertexts,

thereby enabling the server to identify and eliminate duplicates of encrypted chunks. File and Key Recipes, essential for restoration purposes, incorporate the chunk fingerprint and encryption key, respectively, corresponding to each encrypted chunk. These File and Key Recipes are stored on the server. Given the sensitive nature of the encryption keys contained within the File and Key Recipes, they must be secured. This is commonly achieved by encrypting the keys using a client-specific key.

2.2.1 Challenges in Encrypted Deduplication

While Convergent Encryption facilitates deduplication, it introduces vulnerabilities:

Brute-Force Attacks

An attacker can guess the content of a chunk, derive its encryption key, and check for its presence in the storage system. *Message-Locked Encryption* (MLE)[8] formalises the security and vulnerabilities of Convergent Encryption. *DupLESS*[38], or *Server-aided MLE* enhances security by involving a key server in the key derivation process, preventing attackers from deriving keys solely from chunk content.

Side-Channel Attacks

Data privacy can be compromised if deduplication responses reveal the presence of specific data [34]. Attackers can infer whether a file has been stored by observing the answers to deduplication queries. To mitigate this, methods such as the *Random Threshold Scheme*[34], *Gateway Proxy*[35], and *Two-Phase Deduplication* [46, 44] have been proposed. These approaches eliminate cross-user deduplication on the client side or obscure deduplication status to protect against information leakage.

Frequency Analysis Attacks

Deterministic encryption schemes can expose the frequency distribution of plaintext chunks, making them vulnerable to frequency analysis. Techniques like *MinHash Encryption*[45] and *Tunable Encrypted Deduplication* (TED)[98] have been introduced to obfuscate frequency distributions, balancing storage efficiency with data confidentiality.

2.3 Current Challenges and Limitations

As cloud backup services expanded, several challenges emerged:

2.3.1 The Disk Bottleneck

As the adoption of cloud backup increased, backup workloads expanded significantly in volume. This growth resulted from the rising amount of data to be backed up by clients, longer backup timeframes, and the expectation for servers to concurrently manage the backup streams of many clients.

As these workloads grew, they approached the practical limit of unique data identification for a given chunk size using memory-based indexes. One approach to reducing the size of the fingerprint index is to use a larger chunk size [82]. However, these larger chunk sizes generate a reduced deduplication ratio, increasing server data storage requirements.

Early systems had to resort to keeping the hash-based chunk index on disk, but quickly it was realised the random seek access pattern slowed down throughput significantly. From here, we see many papers devoted to increasing throughput in encrypted deduplication systems, as surveyed in Shin et al. [73] and Xia et al. [88]. Most of the designs take advantage of the similarity between successive backups, termed "locality". The locality of backup workloads enabled the construction of lists used to pre-load targeted sets of fingerprints built from previous backups [109, 92, 57]. Or it

enabled segmenting the index so that targeted portions can be loaded as in Min et al. [59]. Other approaches use resemblance techniques [90, 104, 5, 89], which usually do not produce exact deduplication. We first examine some influential exact techniques, followed by important near-exact techniques.

2.3.2 Resource Contention

Resource contention is a critical issue for systems with high volume and/or many concurrent clients. An early approach by Douglis et al. [24] grouped clients with common backup requirements together on specific backup disks to maximise the amount of duplicate data that can be eliminated. A common approach is where chunks are grouped into a "superchunk", and this superchunk is used to route groups of chunks to specific servers for processing (e.g. [29, 107, 77, 28, 22, 54]).

Another approach focuses on reducing resource contention on a single server. Using non-blocking hashing and queues can help reduce resource contention in deduplication, as investigated in Feldman et al. [26]. Sorted Deduplication by Kaiser et al. [36] presented a significant reduction in resource contention for servers deduplicating 1000's of clients concurrently. They deduplicate many clients in a single pass through a sorted CFP index. We discuss Sorted Deduplication further in Section 2.6.

2.3.3 Metadata Storage

Over time, saved File and Key Recipes stored on the server can accumulate a considerable volume and, in long backups, may rival the volume of the stored client data. Metadedup [44] introduced an innovative method of securely deduplicating these recipes, significantly reducing storage requirements for long-running backup workloads. We discuss Metadedup further in Section 2.5 below.

An encrypted deduplication technique that doesn't use MLE and so does not suffer from the large accumulation of encryption keys was introduced by Yang et al.

2022 [97]. They call MLE based encryption "deduplication-after-encryption" (DaE), and they contrast their method as "deduplication-before-encryption" (DbE) on the server. This new technique requires Intel-SGX hardware, and data must pass through a "space-constrained SGX enclave" before being encrypted and saved to disk. This technique does not have to manage a key per chunk metadata but currently doesn't scale to large datasets. In this thesis, we focus on DaE techniques.

2.4 Representative Designs in Encrypted Deduplication

This section examines key designs that have significantly influenced the field, discussing their innovations and limitations.

2.4.1 Reducing Index Size with Segments

To address the trade-off between smaller chunk sizes (which improve deduplication effectiveness) and increased metadata overhead, some approaches group chunks into segments.

2.4.2 Fingerdiff

"Improving duplicate elimination in storage systems" by Bobbarjung et al. [12] introduces Fingerdiff. By default, it generates large chunks as sequences of smaller "subchunks", gathering up to *max_scs* subchunks into a single, large chunk. Fingerdiff's large chunks are similar to SCAIL's segments, which also comprise sequences of chunks up to a given segment size. In "regions of change", smaller chunks will be generated. Fingerdiff identifies these regions on the client using a locally maintained index of previously stored subchunks. The client first concatenates subchunks into chunks, which are then uploaded to the server to be stored.

On the first backup, Fingerdiff creates very large chunks made up of many subchunks and builds a local index of the CFPs of the subchunks. On subsequent backups,

it uses the index to detect regions of contiguous regions of new data. These are concatenated to make new chunks to be uploaded. References are created for the surrounding regions of unchanged subchunks stored on the server. Each of these references consists of the length of the region and its offset into a previously saved chunk.

Fingerdiff focussed on decreasing the average subchunk size to maximise duplicate elimination, exploring average subchunk sizes of 2 KB and below and very large chunks with up to 32,000 subchunks. SCAIL on the other hand, makes use of medium-sized chunks (8 KiB), maintains the exact duplicate elimination enabled by this chunk size, and focuses on reducing memory requirements to increase total server storage capacity.

Note that to operate efficiently, Fingerdiff requires a client to maintain the state of subchunks that have been saved previously to query this information on a per-chunk basis while building its "smaller chunks" in regions of change. In contrast, SCAIL's client is much less compute-intensive because it is stateless and doesn't need to determine the previously-saved status of chunks. Also, Fingerdiff assumed a high level of trust between the client and the server and did not protect against side-channel attacks or provide data privacy, as SCAIL does. Chunk size may also vary across backups. Changed chunk sizes make the chunk lookup index less efficient since the index will contain multiple descriptions of the same data region, each with a different chunk size.

2.4.3 Bimodal

"Bimodal content defined chunking for backup streams", by Kruus et al. [43] focuses on content-defined chunking algorithms for data deduplication. It's called a "bimodal chunking algorithm" because it decides whether to emit a small or large chunk at any given point in the input stream, unlike the baseline content-defined chunking (CDC) methods. There are two main approaches to building chunks for deduplication:

- **Breaking-Apart Algorithm:** This approach starts by chunking data into large

chunks. It identifies regions of new content (changes) and then re-chunks data near the boundaries of these change regions at a finer level. This method is flexible but may result in small changes rendering large chunks non-duplicate.

- **Chunk Amalgamation Algorithm:** In this approach, a smaller chunking size is initially used to generate the chunks, which may subsequently combined into larger ones. These larger chunks can be created by combining a fixed number of small chunks or using a CDC approach to create variable-sized chunks, depending on which algorithm is used. The goal is to identify and emit duplicate data efficiently.

Both approaches aim to achieve cross-user duplicate elimination and assume efficient querying of an index of previously saved chunks. They rely on different strategies to determine chunk boundaries by breaking apart large chunks or amalgamating small ones. The choice of approach and algorithm parameters can impact the average chunk size and deduplication efficiency, especially when dealing with regions that have transitions between duplicate and non-duplicate data.

It is worth noting that the bimodal chunking algorithms do not require storing information about finer-grained ”subchunks” and can work with any index capable of answering whether a chunk with a given fingerprint has already been stored.

The decision-making process in ”Bimodal” for whether to append another amalgamated chunk or start a new one can be complex. It primarily depends on the specific algorithm variant, such as the ”k-fixed” or ”k-var” approaches, as described below.

1. **Current Chunk Evaluation:** As the algorithm processes the input data stream, it evaluates the content of the current chunk it’s working with. This chunk could be a small chunk generated by the initial chunking algorithm or a part of a potential larger chunk.

2. **Duplicate Check:** The algorithm checks whether the content of the current chunk

matches any previously encountered chunks, especially large ones. If a match is found, it indicates a duplicate chunk.

3. **Decision Criteria (k-Fixed Algorithm):** In the ”k-fixed” algorithm variant, the decision is based on whether the current chunk can be appended to an existing amalgamated chunk or if it should start a new amalgamated chunk. Here are the criteria:

    - If the current chunk is a duplicate (already seen before), it’s typically appended to the existing amalgamated chunk.
    - If the current chunk is not a duplicate and follows a region of duplicate data (indicating a transition), it may start a new amalgamated chunk. This is done to minimise inefficiencies around change regions.

4. **Decision Criteria (k-Variable Algorithm):** The ”k-var” variant introduces more flexibility by allowing variable-sized big chunks to be queried at every possible small chunk position during the decision-making process. The criteria are as follows:

    - Like ”k-fixed,” if the current chunk is a duplicate, it’s often appended to an existing amalgamated chunk.
    - If the current chunk is not a duplicate and follows a region of duplicate data, it might still be appended to an existing chunk, but the algorithm can query for a bigger chunk at multiple positions before making the decision.

5. **Amalgamation Process:** If the decision is to append to an existing amalgamated chunk, the algorithm combines the current chunk with the previous one, effectively increasing the size of the chunk. If the decision is to start a new amalgamated chunk, the algorithm creates a new one and adds the current chunk.

6. **Output:** The final decision impacts the chunks emitted as part of the deduplication process. Appending to an existing chunk results in larger chunks being emitted, whereas starting a new chunk leads to smaller ones.

7. **Iteration:** The algorithm continues to iterate through the data stream, similarly evaluating each chunk, making decisions based on whether the content is a duplicate, follows a region of duplicate data, or is a fresh data segment.

The choice of the algorithm variant (k-fixed or k-var) and the specific parameters used can influence the behaviour of "Bimodal" regarding chunk size and deduplication efficiency. These algorithms are designed to strike a balance between minimizing storage overhead and maximizing deduplication effectiveness in the context of changing data streams.

SCAIL Comparison. In the Amalgamation k-var approach (which is closest to SCAIL's segment technique), Bimodal requires not only the knowledge of whether each chunk has been saved previously (i.e. is a duplicate), but also whether the amalgamation of chunks so far has been saved previously. SCAIL does not perform lookups at this level, only at the segment level, and assumes that all chunks of previously saved segments have been saved previously (they are duplicates).

2.4.4 Algorithm Terminology and Notations

Understanding the terminology and notations used in deduplication algorithms is essential. Table 2.2 and Table 2.1 provide a legend for notations and a list of variables used in algorithms.

Notation/Function	Description
$(x, y, \dots)$	Ordered list of elements
$index[key]$	Access $value$ stored at key in $index$
$list[index]$	Access the element at $index$ offset in $list$
$\min(list)$	Returns smallest value in $list$
$a \mod b$	Returns the remainder after division of a by b
$\|a\|$	Returns the length of a in bytes
$\text{Hash}(c)$	A cryptographic hash function applied to c
$\text{Encrypt}(a, key)$	Symmetric encryption of a with key key
$\text{Decrypt}(a, key)$	Symmetric decryption of a with key key

Figure 2.2: Legend for Notation and Functions Used in Algorithms

Name	Description
$\mathbf{C_{size}}$	Approximate, average **C**hunk size of each chunk in a backup.
$\mathbf{S_{size}}$	Approximate, average size of the sum of the chunks in a **S**egment.
FI	**F**ile **I**ndex storing file recipes and metachunk data by file name.
C	**C**hunks $(c_1, c_2, \dots, c_n)$ generated by Context-Driven Chunking (CDC).
CK	**C**hunk encryption **K**eys $(ck_1, ck_2, \dots, ck_n)$, hashes of chunks **C**.
EC	**E**ncrypted **C**hunks $(ec_1, ec_2, \dots, ec_n)$, after encryption with **CK**.
CFP	**C**hunk **F**inger**P**rints $(cfp_1, cfp_2, \dots, cfp_n)$, hashes of **EC**.
MD	**M**eta**D**ata for chunks $(md_1, md_2, \dots, md_n)$, each having (cfp, length, key).
MC	**M**eta**C**hunks $(mc_1, mc_2, \dots, mc_m)$, grouped chunk metadata.
PR	**P**laintext metachunk **R**ecipes $(pr_1, pr_2, \dots, pr_m)$, extracted from metachunks.
MK	**M**etachunk encryption **K**eys $(mk_1, mk_2, \dots, mk_m)$, hashes of metachunks **MC**.
MFP	**M**etachunk **F**inger**P**rints $(mfp_1, mfp_2, \dots, mfp_m)$, hashes of **EMC**.
EMC	**E**ncrypted **M**eta**C**hunks $(emc_1, emc_2, \dots, emc_m)$, after encryption with **MK**.
PEMC	**P**artially **E**ncrypted **M**eta**C**hunks $(pemc_1, pemc_2, \dots, pemc_m)$, each (pr_j, emc_j).
EMK	**E**ncrypted metachunk **K**ey recipe, client encryption of **MK**.
MM	**M**issing **M**etachunks $(mm_1, mm_2, \dots, mm_m)$, a list of **MFP**s.
RFP	**R**esemblance **F**inger**P**rints $(rfp_1, rfp_2, \dots, rfp_m)$, smallest CFP in each **PR**.
RR	**R**esemblance **R**ecipes $(rr_1, rr_2, \dots, rr_m)$, groups (MFPs, PRs, and RFPs).
MR	**M**issing **R**esemblance, missing metachunks/chunks after resemblance.
MI	**M**etachunk **I**ndex mapping metachunk fingerprints to storage container IDs.
RI	**R**esemblance **I**ndex $(rfp, CIDList)$, storage locations for similar segments.
SCI	**S**orted **C**hunk **I**ndex, sorted, on-disk (cfp, cid) chunk storage locations.
DI	**D**uplicate **I**ndex with storage locations for previously uploaded chunks.
CI	**C**ross-user **I**ndex of chunks uploaded by multiple users in one batch.

Table 2.1: Variables Used in Deduplication Algorithms

2.5 Metadedup

"Metadedup: Deduplicating Metadata in Encrypted Deduplication via Indirection", by Li et al. [44], introduced a system that deduplicates metadata to reduce its storage impact, especially in encrypted deduplication.

They examined the additional overhead introduced by key management in Message-Locked Encryption (MLE)-based deduplication compared to plain deduplication. For their analysis, they assumed 30 bytes (B) of metadata per chunk (Chunk Fingerprint (CFP), length, and other information), along with a 32B AES-256 encryption key. With these parameters, for a logical data size of 50TB and a Duplication Elimination Ratio (DER) of 50, the storage overhead due to metadata in plain deduplication was calculated to be 18.7%, which increases to 38.2% in the case of encrypted deduplication.

The high metadata overhead in encrypted deduplication was also experimentally verified after backup operations on the datasets in their evaluation (see dataset descriptions below in Subsection 2.5.10). File and Key Recipes in the FSL dataset accumulate to 369.7GB, which is 86% of the physical (deduplicated) data size of 431.9GB. The metadata in the VM dataset accumulated to 615.2GB, over 3.6 times the 168.2GB of physical data.

The authors of Metadedup note that backup workloads have data redundancy and observed that File Recipes and Key Recipes should also exhibit corresponding redundancy.

2.5.1 System Model

Metadedup follows a client-server model designed to reduce metadata storage overhead in encrypted deduplication systems as shown in **Metadedup Write Operation** on page 50. A table of notations used in the algorithm figures for this thesis is on page 48, and a list of variables used is on page 48. The client processes files by breaking them into chunks, encrypting each chunk using MLE, and then aggregating the metadata

Metadedup Write Operation

Client input: target file name, client's master key key,
segment size S_{size}, chunk size C_{size}
Split file into chunks of average approximate size is C_{size}: $C = (c_1, c_2, \ldots, c_n)$
Perform Message Locked Encryption (MLE) on chunks C, generating
$CK = (ck_1, ck_2, \ldots, ck_n)$, the chunk encryption keys,
$EC = (ec_1, ec_2, \ldots, ec_n)$, the encrypted chunks, and
$CFP = (cfp_1, cfp_2, \ldots, cfp_n)$ are the chunk fingerprints,
where, for i in $[1, n]$:
$ck_i = \text{Hash}(c_i)$, is the chunk key,
$ec_i = \text{Encrypt}(c_i,\ ck_i)$, is the encrypted chunk, and
$cfp_i = \text{Hash}(ec_i)$ is the fingerprint of the encrypted chunk.
Gather chunk metadata $MD = (md_1, md_2, \ldots, md_n)$, where, for i in $[1, n]$:
$md_i = (cfp_i, len_i, k_i)$ is the chunk metadata,
cfp_i is the fingerprint (hash) of the encrypted chunk ec_i, and
len_i is $|c_i|$, the length of chunk c_i.
Calculate segment divisor: $D = \frac{S_{size}}{C_{size}}$
Group MD into metachunks $MC = (mc_1, mc_2, \ldots, mc_m)$, where, for j in $[1, m]$:
mc_j is a consecutive subsequence of MD entries, and
the sum of chunk lengths in mc_j is between ½ and twice S_{size}, and
only the final CFP in mc_j satisfies $(cfp \mod D) = 0$.
Perform Message Locked Encryption (MLE) on metachunks MC, generating
$MK = (mk_1, mk_2, \ldots, mk_m)$, the metachunk encryption keys,
$EMC = (emc_1, emc_2, \ldots, emc_m)$, the encrypted metachunks, and
$MFP = (mfp_1, mfp_2, \ldots, mfp_m)$ are the metachunk fingerprints,
where, for i in $[1, m]$
$mk_j = \text{Hash}(mc_i)$ is the metachunk key,
$emc_j = \text{Encrypt}(mc_j,\ mk_j)$, is the encrypted metachunk, and
$mfp_j = \text{Hash}(emc_j)$ is the metachunk fingerprint.
Encrypt metachunk key recipe: $EMK = \text{Encrypt}(MK,\ key)$
Send: file name, metachunk file recipe MFP, encrypted key recipe EMK,
encrypted chunks EC, and encrypted metachunks EMC to the server

Server input: fingerprint index FPI, file index FI
Receive: file name, metachunk file recipe MFP, encrypted key recipe EMK,
encrypted chunks EC, and encrypted metachunks EMC
Store unique EC and EMC using FPI, noting storage locations
Store MFP, EMK in FI by file name

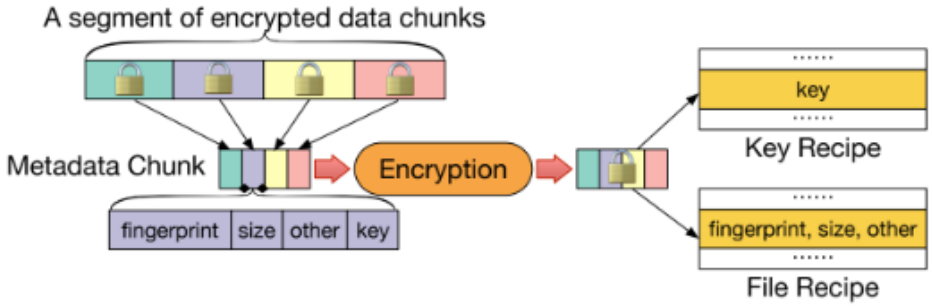


Figure 2.3: Illustration of Metadedup's metachunk construction process. Metadata from segment chunks are aggregated into a Metadata Chunk, termed a *metachunk*, which is encrypted, generating the hash key, encrypted metachunk, and metachunk fingerprint. File Recipes comprise lists of metachunk fingerprints, while the Key Recipes contain the corresponding encryption keys. Chart reproduced from Figure 3 in [44].

of these chunks into larger units which we call *metadata chunks*. The metachunks are themselves encrypted and deduplicated.

To protect against side-channel attacks, where malicious clients might exploit deduplication responses to infer the presence of specific data stored by other clients [34, 33], Metadedup employs a two-phase deduplication process. In the first phase, client-side deduplication is performed only within the scope of a single client's data; no cross-client deduplication is conducted at the client side. In the second phase, the server performs deduplication across all clients but does not reveal deduplication status to individual clients. This design choice prevents adversaries from leveraging deduplication responses to gain unauthorised information about data stored by other clients.

2.5.2 Building Encrypted Chunks

The client uses a Content-Defined Chunking (CDC) algorithm to segment files into chunks of a target average size. Each chunk is then encrypted using MLE, where the encryption key is derived from the content of the chunk itself. The encryption of a chunk c generates:

1. The encryption key ($ck = \text{Hash}(c)$).
2. The length of the chunk ($len = |c|$).

3. The encrypted chunk data ($ec = \text{Encrypt}(c, ck)$).
4. The CFP, which is the hash of the encrypted chunk ($cfp = \text{Hash}(ec)$).

2.5.3 Building Encrypted Metachunks

The client groups the metadata of encrypted chunks into *metadata chunks*, which we refer to as *metachunks*. A CDC algorithm is applied to the sequence of CFPs to determine segment boundaries, aiming for a target average segment size (e.g., 2 MiB). The aggregated metadata of the chunks within a segment forms a metachunk. Each metachunk includes the CFP, length, and encryption key of its constituent chunks.

The metachunk is then encrypted using MLE, where the encryption of a metachunk mc generates:

1. The metachunk encryption key ($mk = \text{Hash}(mc)$).
2. The length of the metachunk (len).
3. The encrypted metachunk data ($emc = \text{Encrypt}(mc, mk)$).
4. The Metachunk Fingerprint (MFP), the hash of emc ($mfp = \text{Hash}(emc)$).

2.5.4 File Recipes and Key Recipes

For each file, the client constructs a File Recipe comprising the sequence of MFPs corresponding to the metachunks of the file. The client also creates a Key Recipe containing the metachunk encryption keys, which is encrypted using a client-specific master key to protect the keys.

2.5.5 Backup and Metadata Deduplication

The client performs intra-user (within-client) deduplication by comparing the MFPs of the current backup with those from previous backups of the same client. Metachunks identified as new (not previously stored by the client) are scheduled for upload.

To avoid side-channel attacks, the client does not perform cross-client deduplication. Instead, cross-client deduplication is performed on the server, but without revealing deduplication status to clients. When metachunks and chunks are uploaded, the server identifies duplicates across all clients and stores only unique data, updating ownership information accordingly.

2.5.6 Restore Operations

To restore a file, as shown in **Metadedup Restore Operation** on page 54, the client sends the file name to the server (Lines 1-2). The server retrieves the File Recipe and Key Recipe from the File Index and retrieves the file's encrypted metachunks based on the MFPs in the File Recipe (Lines 3-6). The server sends the File Recipe, Key Recipe and the encrypted metachunks for the file back to the client (Line 7). The client first decrypts the encrypted metachunks, which gives it the CFPs for the chunks of the metachunk (Lines 8-12). The client requests, and the server returns these encrypted chunks (Lines 13-17). The client uses the keys for the chunks extracted from the metachunks to decrypt the chunks and assemble them into the restored target file (Lines 18-22).

The client then decrypts each metachunk to retrieve the metadata of the constituent chunks, including their CFPs and chunk encryption keys. With this information, the client requests the necessary encrypted chunks from the server based on the CFPs. Finally, the client decrypts the chunks using their encryption keys and reconstructs the original file.

Metadedup Restore Operation

1: **Client Input:** target file name, client's master key key
2: Request file metadata from the server using the target file name

3: **Server input:** fingerprint index FPI, file index FI
4: Receive: target file name from the client
5: Retrieve file recipe MFP, EMK from FI based on the file name
6: Retrieve encrypted metachunks EMC based on MFP using FPI
7: Send: MFP, EMK and EMC to the client

8: **Client input:** client's master key key
9: Receive: MFP, EMK and EMC
10: Decrypt key recipe: $MK = \text{Decrypt}(EMK,\ key)$
11: Decrypt metachunks EMC with keys MK, generating
$MC = (mc_1, mc_2, \ldots, mc_m)$, the decrypted metachunks
where, for j in $[1, m]$
$mc_j = \text{Decrypt}(emc_j, mk_j)$.
12: Extract all chunk fingerprints $CFP = (cfp_1, cfp_2, \ldots, cfp_n)$ from all MC
13: Request encrypted chunks EC from the server using chunk fingerprints CFP

14: **Server input:** fingerprint index FPI
15: Receive: chunk fingerprints CFP from the client
16: Retrieve encrypted chunks EC based on CFP using FPI
17: Send: EC to the client

18: **Client input:** metachunks MC
19: Receive: encrypted chunks EC
20: Extract chunk keys $CK = (ck_1, ck_2, \ldots, ck_n)$ from MC
21: Decrypt chunks EC with keys CK, generating
$C = (c_1, c_2, \ldots, c_n)$, the decrypted chunks
where, for i in $[1, n]$:
$c_i = \text{Decrypt}(ec_i, ck_i)$.
22: Assemble chunks (c_i) in order to reconstruct the original file

2.5.7 Security Analysis

Metadedup ensures data confidentiality and integrity and assumes a threat model with an honest-but-curious server who follows the protocol correctly but attempts to gain unauthorised access to data and metadata. An adversary may have the following capabilities:

1. **Server Compromise**: The adversary may gain access to all encrypted data and metadata stored on the server, including the fingerprint index, file recipes, key recipes, encrypted chunks, and encrypted metachunks.

2. **Client Compromise**: The adversary may compromise certain clients, obtaining their original data, metadata, and master keys. The goal is to infer data or metadata belonging to other clients, which the compromised clients are not authorised to access.

Confidentiality of Data Chunks

Each data chunk is encrypted using MLE, with the encryption key derived from the chunk's content. To protect against brute-force attacks on *predictable* content (e.g., widely known, easily guessed or structured with a predictable value or pattern), Metadedup can adopt the DupLESS approach [38], where clients obtain per-chunk encryption keys from a key server via an Oblivious Pseudo-Random Function protocol. This method prevents attackers from deriving encryption keys solely from chunk content, enhancing security even for predictable data.

Metachunk Confidentiality

Under traditional MLE without server assistance, an adversary could derive metadata (e.g., keys) directly from data chunks and may attempt to construct metadata chunks arbitrarily. Since Metadedup protects metadata chunks using MLE, the adver-

sary could, in theory, launch an offline brute-force attack to infer the original contents of target metadata chunks.

However, the authors of Metadedup argue that such an attack is computationally infeasible due to the enormous computational cost. Each metadata chunk in Metadedup consists of the metadata of multiple encrypted data chunks. To perform an offline brute-force attack, the adversary would need to consider all possible *ordered sequences* of encrypted data chunks to construct potential metadata chunks, because the order of chunks affects the encryption of the metadata. The number of such sequences is prohibitively large.

The following bullet points summarise Metadedup's analysis from Section IV-D. Let:

- n be the total number of distinct data chunks.
- c be the average number of data chunks in a segment.
- T_{hash} be the time to perform a hash operation.
- T_{enc} be the time to perform an encryption operation.
- $N = \frac{n!}{(n-c)!}$ be total number of possible *permutations* (ordered sequences).
- $T_{\text{attack}} = N \times (3T_{\text{hash}} + 2T_{\text{enc}})$ be a single attack time.
- $T_{\text{hash}} = 37\,\mu\text{s}$, $T_{\text{enc}} = 48\,\mu\text{s}$ be typical hash and encryption times.
- $T_{\text{attack}} \geq (3T_{\text{hash}} + 2T_{\text{enc}}) \times c! \approx 7.94 \times 10^{211}$ be the total attack time in seconds.
- 10^{204} be the total attack time in years.

Thus the Metadedup authors conclude that the attack is infeasible.

However, we note that this analysis assumes that the adversary cannot reuse computations from previous chunk encryptions. In reality, an attacker could cache the encrypted chunks and their metadata, reducing the number of calculations required for individual chunk encryptions. Since the maximum number of distinct data chunks is n, the adversary needs to perform at most n encryption operations to obtain all possible encrypted chunks.

Even with this optimisation, the adversary still faces an impractically large number

of permutations when constructing metadata chunks. The total number of possible metadata chunks remains combinatorially large due to the need to consider all possible permutations of c encrypted chunks:

$N = \frac{n!}{(n-c)!}$.

This calculation of N from Metadedup's security analysis assumes that all chunks within a metachunk are distinct, which does not have to be the case; repeated chunks in a metachunk are valid. However, this assumption simplifies the analysis, and it represents a conservative estimate of the number of permutations. While a more accurate analysis would consider an even larger number of permutations, the result would remain computationally infeasible for any practical attack scenario.

Handling Small Files Small files require careful handling because a metachunk with only a few constituent chunks significantly reduces the number of permutations N, making brute-force attacks feasible. For example, if a metachunk consists of only a few chunks, the adversary can more easily enumerate all possible permutations.

To mitigate this vulnerability, it would be possible for Metadedup to treat incoming file data as a continuous stream, rather than creating a single metachunk for files smaller than the segment size. By integrating small files into larger metachunks that include data from multiple files, or by padding metachunks to a minimum size, the system maintains a high combinatorial complexity, keeping N large enough to prevent feasible brute-force attacks. Therefore, the system remains secure when files are streamed, and small files do not compromise the overall security of metadata confidentiality.

Protection Against Side-Channel Attacks

Metadedup mitigates side-channel attacks that exploit deduplication responses to infer the existence of specific data [34]. Since client-side deduplication is performed only within a single client's data, and cross-client deduplication occurs solely on the server

without revealing deduplication status to clients, adversaries cannot use deduplication responses to learn about data stored by other clients.

Resilience to Compromised Clients

If the adversary compromises certain clients and obtains their data, metadata, and master keys, they cannot decrypt data or metadata belonging to other clients. Each client's metachunk decryption keys are protected by encryption keys specific to that client. Being unable to decrypt the metachunks means the client cannot access the chunk encryption keys, preventing unauthorised access to chunk data.

Integrity and Availability

The authors state that Metadedup can be integrated with existing integrity verification schemes, such as data auditing protocols [6, 21]. These mechanisms protect against malicious modifications or deletions by the server, ensuring data integrity.

Availability and resilience can be supported through the mechanism of deduplication-aware secret sharing with CDStore [46]. In fact, the Metadedup evaluation prototype is built from CDStore.

2.5.8 Limitations

Metadedup achieves significant storage savings for long-running or high-volume backup datasets, but its design is limited by the use of a memory-based index that holds both CFPs and MFPs. While MFPs only add about 1.9% to the total index size, the majority of the memory requirements come from the far more numerous CFPs. This fundamentally limits the scalability of Metadedup to cases where the entire index can reside in the main memory.

The reliance on a memory-based index means that the volume of backup data Metadedup can handle is ultimately capped by available RAM. Unlike SCAIL, which

maintains only metachunk fingerprints in memory, allowing for a much smaller index size, Metadedup must store individual chunk fingerprints. As the dataset grows, the memory consumption grows proportionally, eventually becoming infeasible to manage entirely in RAM.

Using a disk-based index for Metadedup would incur a significant slowdown, as duplicate-lookup queries require high-speed access to fingerprints. Disk I/O latency would severely impact system throughput, making the deduplication process impractically slow. This trade-off between memory and scalability represents a limitation of the Metadedup approach.

At the petabyte scale of deduplication, Metadedup does not have low memory requirements and would be required to use a disk-based index, so it is not capable of fast, petabyte-scale deduplication.

2.5.9 Evaluation

Metadedup's microbenchmark tests showed that performance slowed as backed-up synthetic data increased from 1 to 20GB, with save and restore speeds dropping from 63.3MB/s and 87.9MB/s to 45.5MB/s and 70.7MB/s, respectively. Metadedup also adds a performance overhead compared to CDStore[46], with save operations showing an increased processing time of 6.19% and restore operations an increase of 23.23%. Despite this, the benefits of metadata deduplication were evident in trace-driven simulations using the FSL and VM datasets, described next.

2.5.10 Metadedup Datasets

1. **FSL**: A total of 115 selected backup days from the home directories of students in the File systems and Storage Lab. Our work uses this identical dataset in its evaluations; for further details, see Subsection 3.12.1.

2. **VM**: A private dataset consisting of 156 Virtual Machine snapshots collected by

the Metadedup authors over 26 backup days.

Both datasets were evaluated with segment sizes of 512KB, 1MB, 2MB, and 4MB, all resulting in significant reductions in metadata storage. The FSL dataset showed metadata storage reductions exceeding 97% for each segment size. Metadedup 's index size overhead (from added MFPs) for the FSL dataset reduced as the segment size increased, from 1.9% for 512KiB segments to 0.33% for 4MiB segments. Similarly, the VM dataset showed substantial metadata storage reductions (between 93% and 95%) for each of the segment sizes, with a maximum index size overhead of 1.9% at 512KiB down to 0.39% with 4MiB. Evaluation results showed that additional compression of metachunks did not yield any significant improvements in storage efficiency; hence, Metadedup did not employ metachunk compression.

2.5.11 Metadedup Summary

Metadedup reduces server storage requirements more effectively than any other method in the literature that we are aware of. In our experiments, it reduces overall storage by up to 44% (see Subsection 6.4.1). This significant reduction is achieved by duplicating the metadata stored with each backup. Metadedup aggregates chunk metadata into metachunks and performs deduplication at both the chunk and metachunk levels. This approach significantly reduces storage volume requirements in backup scenarios, making Metadedup state-of-the-art for minimizing storage needs. To the best of our knowledge, no other system provides this level of additional savings beyond conventional chunk deduplication.

By employing techniques such as Message-Locked Encryption (MLE) and the DupLESS approach for encryption, and restricting client-side deduplication to within-client data, Metadedup ensures strong confidentiality and integrity guarantees. It mitigates potential attacks, including offline brute-force and side-channel attacks, rendering them computationally infeasible or ineffective.

However, special attention must be paid to the handling of small files that may fit into a single metachunk, as this could reduce the complexity of brute-force attacks. By processing data as a continuous stream of all files to be backed up, the system can maintain high levels of data security, even for small files.

While Metadedup achieves unmatched efficiency for storage reduction, its scalability is limited by its reliance on the chunk fingerprint-based deduplication index. If we assume 256 GB of memory for the deduplication server, and each entry in the index requires 30 B (20 B SHA-1 hash, 6 B container ID, other metadata), representing an 8 KB chunk, this would enable deduplication of up to 69 TB of data. Therefore, a dataset with 100 TB of unique data is a realistic upper limit for its capacity.

2.6 Sorted Deduplication

"Sorted deduplication: How to process thousands of backup streams" by Kaiser et al. [36] introduces a "new exact deduplication approach designed for processing thousands of backup streams at the same time on the same fingerprint index". No other system that we know about purports to support even a fraction of this many streams.

Traditional deduplication systems were designed to process a limited number of backup streams. When traditional systems were tasked with processing ever-increasing numbers of clients, it led to non-contiguous disk accesses and performance bottlenecks due to resource competition among streams. In [36], the authors introduced a novel, exact deduplication approach capable of handling thousands of backup streams simultaneously without causing memory contention or heavy disk I/O.

The new method, Sorted Deduplication, enhances performance by sorting fingerprints, ensuring sequential disk access patterns and reducing I/O operations significantly. The proposed implementation of the method called Sorted Chunk Indexing (SCI) processes backup streams in sorted order, leading to high index locality and efficient disk I/O use. The paper shows that their method outperforms existing systems

like the Data Domain File System [109] (DDFS) and Sparse Indexing [49] (SI) by a considerable margin in terms of I/O reduction and memory usage.

2.6.1 Background

Most deduplication systems rely on a chunk index to identify redundancies within the data, where a unique fingerprint represents each data chunk. Traditionally, the primary performance bottleneck has been the chunk index lookups. This index is often too extensive to fit entirely in memory, leading to reliance on disk storage, which markedly reduces throughput. This is due to two main reasons: the process slows to disk access speeds, which are orders of magnitude slower than memory access speeds, and the nature of hash-based indexes that necessitate random seeks.

Random seeks are less efficient than sequential reads due to the mechanical movements required in hard disk drives (HDDs) to position the read/write head at the appropriate location on the disk platter. Even with solid-state drives (SSDs), which have no mechanical parts, random access is slower than sequential access due to how data is organised and accessed in the storage architecture.

To mitigate against these issues, previous designs have attempted to reduce the size of the chunk index to fit into the main memory, but this reduction in performance comes at the expense of exact deduplication efficiency. That is, smaller indexes lead to a higher chance of missing duplicate chunks, thus storing redundant data.

Another approach to alleviating the performance hit from disk-based index lookups is to exploit chunk locality, which is where sets of chunks tend to appear together between subsequent backups. This method can generate near-sequential disk access patterns for a single backup stream, enhancing performance significantly. However, this advantage diminishes when multiple streams are processed in parallel. Despite each stream having strong locality, the overall access pattern can degrade to a random one because different streams exhibit different localities. Furthermore, these streams com-

pete for limited memory space, making it challenging to maintain a single effective locality cache for all streams.

In corporate environments, for instance, peak demand for backups can number in the thousands, exacerbating these challenges. Streams processed concurrently during such peak times further complicate the deduplication process, making it imperative to devise a method that can efficiently handle the increased load without substantially compromising performance or deduplication accuracy.

2.6.2 Design

The Sorted Chunk Indexing (SCI) design is centred around creating a uniform locality across all backup streams by processing the streams' Chunk Fingerprint (CFP)s in a single sorted sequence. This is achieved against a single disk-based index that is also held in sorted order, culminating in an optimal sequential disk access pattern.

Server Design The server component of SCI is tasked with identifying and storing new chunks. It is designed to handle multiple streams concurrently, allowing for simultaneous start times across streams, but also accommodating staggered start times. The server uses an LSM tree (a log-structured merge-tree) [65] as the chunk fingerprint index. The leaves of the tree hold chunk fingerprints sequentially and in sorted order. The height of the tree is limited to 2 levels. This LSM tree structure is crucial as it permits the server to process all CFPs within each leaf's range before proceeding to the next, ensuring an efficient and orderly data processing flow. Since the leaves are read-only during this operation, the server can manage parallel stream processing effectively. Moreover, consolidating many CFPs into leaf pages significantly reduces I/O, with the total reads during a deduplication pass being at most equal to the number of leaves.

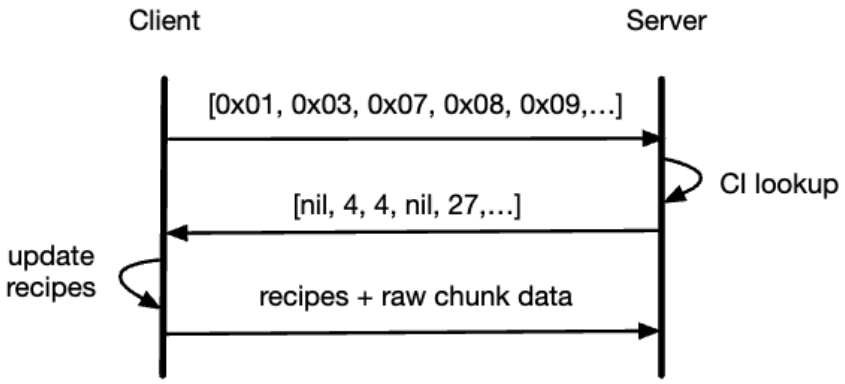


Figure 2.4: Sorted Deduplication's Client-Server Communication. Clients send sorted fingerprints to the server, which returns the container ids for previously saved chunks, and nil for new chunks. The client builds restore recipes and sends these and the new chunk data to the server. This figure is reproduced from Figure 2 in [36].

Client Design On the client side, the primary functions include chunking the backup data and generating CFPs. All clients employ the same chunking and hashing methods to enable cross-client deduplication. After chunk generation, the CFPs are sorted, and any duplicate chunks found locally in the client's data are eliminated. These sorted CFPs are then transmitted to the server, as shown in Figure 2.4, which responds with a correlated list of CIDs indicating the storage location for each chunk or an indicator flagging a new chunk. Returning the CID location to the client is pivotal for the Sorted Deduplication process, as it allows both the client and the server to collaboratively construct *restore* recipes that can be processed during restore operations without requiring a full pass through the disk-based chunk index.

Operational Efficiency and Overhead Sorting the CFPs incurs an overhead on the client side, with the sorting operation bearing a computational complexity of $O(n \log n)$ (n is the client chunk count). Nevertheless, this overhead is justified by the significant reduction in I/O operations on the server side, leading to a more streamlined and efficient deduplication process.

Multi-server Support and Scalability To accommodate scalability demands, SCI also supports a multi-server architecture, either by distributing clients across servers or by dividing the CFP space among the servers. The former approach does not accommodate cross-client chunk deduplication across servers, and so does not perform exact deduplication. The latter approach, which partitions the CFP space, enforces exact deduplication across the system. If new servers are added to the system, the CFP subranges will be modified to evenly distribute the load. These subranges can be further adjusted to conform to the LSM leaf page boundaries. This allows for a straightforward redistribution of the chunk index, thereby simplifying the system's scalability.

2.6.3 Limitations

Despite its efficient design, the authors of Sorted Deduplication acknowledge a limitation in its operational paradigm: even when the number of CFPs to be checked for client-side deduplication is relatively small, a comprehensive pass through the disk-based chunk index is still required. This can be substantially slower than looking up each CFP in a disk-based hash index. Also, Sorted Deduplication is optimised for efficiency rather than privacy and requires a high level of trust between client and server, especially in the area of recipe management. The server must trust that the recipes returned faithfully reflect the CIDs returned in the client-side deduplication query.

Therefore, Sorted Deduplication does not offer data privacy or side-channel protection guarantees. Since it does not deduplicate metadata, it does not offer reduced metadata storage or reduced upload volume. Sorted Deduplication did not publish throughput figures for their evaluation datasets; they only compared the number of I/Os saved vs competing schemes. Therefore cannot estimate their throughput numbers. See the feature comparison in Table 2.2.

2.6.4 Evaluation Results

Sorted Deduplication was compared with Data Domain File System (DDFS) [109] and Sparse Indexing (SI) [49] using two distinct datasets. The first dataset originates from the Johannes Gutenberg University HPC cluster, consisting of 597 streams, including a full backup and 60 days of incremental backups (not all streams participated in all days). This dataset encompasses a logical data volume of 8 TB, which deduplication processes reduced to a physical size of 3.7 TB with 8 KB chunks. The second dataset is derived from Microsoft (MS) traces [58], featuring the incremental backups of 140 randomly selected streams over 33 days, cumulatively amounting to 48.7 TB of logical data, deduplicated using 8 KB chunks to 7.8 TB of physical data.

The deduplication systems in the experiments were configured to manage up to 16 TiB of physical data and allocated 8 GiB of memory. For DDFS, a Bloom filter with a 1% false-positive rate, taking 2.4 GB of memory, was established. The remaining memory was dedicated to a container cache. Containers were set to 4 MB, and allowing for 50% compression of data could hold around 1024 chunks. The SI scheme utilised in-memory sampling of CFPs at a rate of $\frac{1}{16}$, dedicating the remaining memory to a manifest cache capable of storing more than 27,000 manifests. Within SCI, 1 GB is allocated to the Log-Structured Merge Tree (LSM tree), plus 128 MB for one leaf node.

The principal performance metric utilised was the number of Input/Output operations (I/Os) required per 1K (1,000) chunks processed during deduplication, particularly focusing on the systems' performance in a "steady state," therefore excluding the initial backup from the analysis.

Performance Analysis The initial experiment explored duplicate chunk lookup throughput, and involved generating synthetic chunk data and varying sizes of prefabricated client data and chunk indexes. The findings indicated that SCI's throughput increases with the number of clients, the amount of data backed up per client, or a combi-

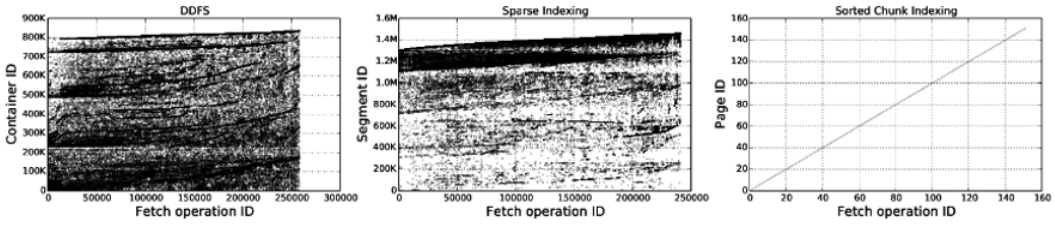


Figure 2.5: Comparison of I/O patterns for 18th backup of the MS dataset, with Data Domain File System (left), Sparse Indexing (middle) and Sorted Chunk Indexing (right). Sorted Chunk Indexing requires only a small number of sequential disk I/Os. This figure is reproduced from Figure 10 in [36].

nation of both. In the specific scenario with 1,024 clients (the largest number evaluated), where each client backed up 1 GB of data, the throughput reached about 5 GB/second. When each client backed up 16 GB of data, the throughput increased to 40 GB/second, and with each client backing up 32 GB of data, the throughput peaked at 70 GB/second.

Although DDFS and SI exhibited less I/O than SCI for the smallest evaluated dataset with 20 client streams, their efficacy dropped considerably with more than 20 clients. In the context of the largest backup datasets, SCI demonstrated superior performance, requiring up to 12 times less I/Os than SI and up to 113 times less I/Os compared to DDFS.

I/O Patterns Figure 2.5 shows the I/O access patterns of the evaluated deduplication strategies during the 18th daily backup generation of the MS dataset. This graphical representation reveals SCI's linear access pattern, in stark contrast to the random fetch patterns exhibited by DDFS and SI, underscoring the limitations of their caching mechanisms in reducing frequent data fetches.

Memory Variation Experiment An experiment to evaluate the impact of available memory, varied from 128 MB to 64 GB, revealed that SCI consistently maintained a low rate (lower is better) of around 0.3 I/Os per 1K chunks across all available memory sizes. SI ranged from 0.36 to 0.64 I/Os per 1K chunk, but missed 25% of duplicates at

128 MB available memory, improving to missing less than 1% with 64 GB. DDFS started at a high 3.5 I/Os per 1K chunks at 8 GB and decreased to 1.4 I/Os per 1K chunks at 64 GB of memory. The experiment showed that Sorted Deduplication requires much less memory than the competing schemes and requires fewer disk I/Os.

2.7 Resemblance Mergence Deduplication (RMD)

"RMD: A Resemblance and Mergence Based Approach for High Performance Deduplication" by Zhang et al. [104] is a near-exact deduplication scheme that uses fixed-size segments in a resemblance technique that reduces memory requirements, enabling relatively fast response times to fingerprint lookup queries for deduplication. It stores the CFPs of all similar segments (segments with the same representative fingerprint) in an *FpBin* file and keeps a count of how many times a CFP has been added. When the number of CFPs in an FpBin file exceeds its capacity, truncation occurs, preferentially removing CFPs that have been added the least frequently. Additionally, CFPs are culled based on age, with older fingerprints being discarded first. To locate the appropriate FpBin file for a segment, or to ascertain if the segment is new, a Bin Address Table (BATable) along with the Dynamic Bloom filter Array (DBA) (facilitating parallel queries of bloom filters) is employed. For new segments, a corresponding FpBin is created and placed into the *FpBinBuffer*, an LRU (Least Recently Used) memory cache. The *RAM-Hit-Table* monitors the presence of FpBins within the FpBinBuffer. If a segment is not new but its FpBin is absent from the FpBinBuffer, the FpBin is retrieved and positioned at the forefront of the buffer. When the FpBinBuffer reaches its capacity, the least recently used FpBin is offloaded to disk.

2.7.1 Security Analysis

RMD effectively is a server-side deduplication system. No mention is made of privacy guarantees, security analysis, threat models, or attack mitigation techniques. In fact,

if the techniques described were used without modification in a client-server cloud-based backup, they would easily be susceptible to side-channel attacks. Attackers could find the minimum-valued fingerprints with a predictable set of chunks (see Section 2.5) and create fixed-sized segments including each of the minimum fingerprints to check for the presence of specific fingerprints in any of the FpBins stored on the server.

2.7.2 Limitations

RMD offers near-exact deduplication but has limitations due to its segment construction method. It generates segments with a fixed number of chunks, which offer no resistance to data shifts. Small, localised changes between backups can cause chunk fingerprints to move from one segment to another. Consequently, the identifying representative fingerprint may change. This phenomenon can adversely affect the effectiveness of deduplication.

RMD employs its RAM-Hit-Table to oversee the FpBinBuffer, aiming to minimise disk I/O when deduplicating chunk fingerprints of a segment. However, under certain circumstances — like starting with an empty buffer, backing up large data volumes, or processing segments not recently in the buffer (as seen when backing up a new user's data) — the system may necessitate a disk I/O for every segment. Additionally, RMD lacks a quick method to efficiently identify and ignore the most common segment type, namely duplicate segments between backups.

RMD also uses an update mechanism in the FpBinBuffer cache. When a new segment is merged into the FpBin of a similar segment in the FpBinBuffer, an update bit is turned on. When updated FpBins are ejected from the FpBinBuffer, they are written to disk.

While RMD is able to perform deduplication at scale with low-memory requirements, it will perform a significant amount of I/O, which would preclude it from per-

forming fast petabyte scale deduplication. It also does not incorporate mechanisms to provide data privacy, or protection against side-channel attacks. It does not deduplicate metadata, and does not perform exact chunk-level deduplication, so it does not offer reduced metadata storage or reduced upload volumes.

2.8 Research Gap and Motivation

The limitations in existing designs, particularly in handling metadata overhead, scalability, and data privacy, motivate the need for new approaches. Combining the metadata deduplication of Metadedup with the scalability and resource efficiency of Sorted Deduplication presents an opportunity to address these challenges.

Feature	Sorted Deduplication	Base	Metadedup	RMD	SCAIL	R-SCAIL
Exact Chunk-level Deduplication	✓	✓	✓		✓	✓
Low Memory Requirements	✓			✓	✓	✓
Fast Petabyte Deduplication	✓				✓	✓
Data Privacy		✓	✓		✓	✓
Side-channel Protection		✓	✓		✓	✓
Reduced Metadata Storage			✓		✓	✓
Reduced Upload Volume			✓		±	✓
Client-side GiB/s Deduplication	-	0.9	0.9	Est.˜5.2	100.9	5.9

Table 2.2: Research gap showing feature support across schemes. Client-side deduplication throughput figures in the bottom row are for the FSL dataset described in Subsection 3.12.1.

Table 2.2 summarises the features of existing schemes and highlights the gaps that

the proposed solutions aim to fill. Achieving low memory requirements, maintaining data privacy, and reducing metadata storage while ensuring high deduplication throughput are key areas of focus.

In our comparison, we include the scheme Base as a baseline traditional deduplication design. The Base scheme is described in Section 3.12 and uses a chunk fingerprint index on the server for chunk deduplication, so it does not have low memory requirements, requiring a disk-based index above 100 TB of deduplicated data, so it cannot perform fast deduplication at petabyte scale. The Base scheme also does not perform segmenting or metadata deduplication, so it does not offer reduced metadata storage or reduced upload volume.

We observe that the SCAIL design creates redundant segment data uploads because client-side deduplication operates at the segment level rather than the chunk level. Therefore, it does not offer the reductions in upload volume from metadata deduplication that Metadedup and R-SCAIL offer. This limitation is offset by significantly faster client-side deduplication throughput.

Regarding the throughput figures in the last line of Table 2.2, these are derived from our evaluation results on the FSL dataset (see Section 6.1). The authors of Sorted Deduplication state that their design is not conducive to high-speed client-side deduplication; thus, we don't include figures for it. We estimate RMD throughput at 5.2 GiB/s from Figure 6 from their paper [104] where they show a throughput of 1.2 million fingerprints/second, and an average chunk size is 4.67KB, giving approximately 5.2 GiB. Although this experiment was against the FSL dataset, the days and set of users differ from that used by Metadedup and SCAIL.

SCAIL provides all of the features except for a partial score on reduced upload volume, since it uploads redundant segment data. This is often offset by reducing upload volume from deduplicated metadata, so we show $\pm$ in Table 2.2. R-SCAIL consistently has reduced upload volumes since it deduplicates metadata, and performs 80% to 97% of perfect client-side deduplication.

2.9 Summary

This chapter has provided a detailed examination of various deduplication approaches, highlighting the ongoing challenge of balancing efficient duplicate elimination with the constraints of chunk size and metadata memory overhead. Through the exploration of historical and contemporary methodologies, including Fingerdiff, Bimodal, Metadedup, Sorted Deduplication, and RMD, we have identified common threads and gaps in the field.

Each method presents unique solutions and compromises, and the discussion underscores a critical area of research in deduplication: finding effective strategies that minimise metadata overhead, ensure scalability, and maintain data privacy. Despite advancements, current solutions still face limitations in addressing these challenges comprehensively.

The next chapter introduces *SCAIL* (Segmented Chunks and Index Locality), a deduplication framework developed to address the gaps identified in previous approaches. SCAIL is designed to optimise deduplication efficiency, reduce metadata storage, and enhance data security by integrating and improving upon the concepts discussed in this chapter.

Chapter 3

SCAIL: Segment Chunks And Index Locality

3.1 Introduction

The contents of this chapter are adapted from our paper: J. Ammons, T. Fenner, and D. Weston, "SCAIL: Encrypted Deduplication With Segment Chunks and Index Locality," in 2022 IEEE International Conference on Networking, Architecture and Storage (NAS), IEEE, 2022, pp. 1–9

This chapter introduces the SCAIL (**S**egment **C**hunks **A**nd **I**ndex **L**ocality) encrypted deduplication system, designed for high-performance fine-grained deduplication for long or large backup workloads in systems required to support many concurrent clients. Specifically, we:

- Scale-up Metadedup's memory-based client-side duplicate detection capacity by requiring only metachunk fingerprints, deferring to server-side processing for chunk level-deduplication. For large datasets, the metachunk fingerprints will easily fit into memory enabling a fast response to client duplicate lookup queries.
- Redesign Sorted Deduplication's Sorted Chunk Indexing (SCI) to support encrypted deduplication under our threat model, allowing fine-grained full-chunk index server-side deduplication that processes many client streams concurrently

while requiring a small number of disk accesses.

- Combine these two systems into a hybrid, two-phase deduplication process which reduces both metadata storage requirements and memory requirements.
- Implement a prototype of a SCAIL system and gather metrics on two trace-based, public, real-world backup datasets.

3.2 System Design

SCAIL employs the client-server model, and we assume communication channels between the client and server are protected and secure. A *client* runs on the user's machine, which processes backup files and uploads the encrypted file data, and its corresponding deduplication metadata. The server maintains two fingerprint indexes: a memory-based index at the segment level for client-side deduplication, and a disk-based bin structure, adapted from Sorted Deduplication's [36] Sorted Chunk Indexing (SCI) design, at the chunk level for server-side deduplication.

Metadedup [44] performs lookup queries for client-side deduplication and handles restore requests at both the metachunk and chunk fingerprint levels, whereas the SCAIL server interacts with clients exclusively at the metachunk fingerprint level, returning a list of "Missing" MFPs. It is worth noting that by not disclosing the upload status of chunks, SCAIL enhances its resistance to side-channel leakage attacks compared to Metadedup. On the other hand, changing a single chunk of a segment in a subsequent backup in SCAIL will cause the entire segment to be uploaded; however, in this case, all but one of the chunks are then discarded since they have already been stored as elements of the previously saved segment.

In addition, answering lookup queries only at the metachunk fingerprint level means SCAIL does not need to store ownership information for chunk fingerprints, only for the far fewer metachunk fingerprints. Ownership of a metachunk implies

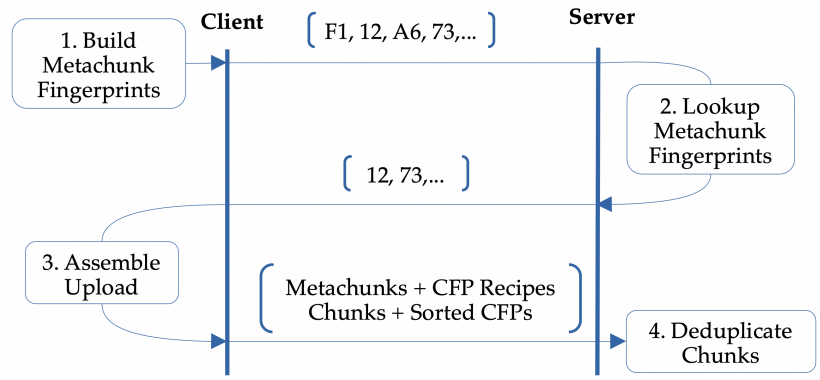


Figure 3.1: SCAIL system data flow in four stages, alternating between Client and Server. Client-side deduplication is performed in stages 1-3.

ownership of its constituent chunks.

For the server-side, chunk-level deduplication, SCAIL uses client-specific data containers to reduce the I/O required to save and load chunks of data. In addition, separate client-specific containers are used to store metachunks.

3.3 SCAIL Algorithm

Figure 3.1 shows an example of SCAIL's encrypted deduplication data flow in four stages, alternating between the client and the server. Client-side deduplication is performed in Stages 1-3, and server-side deduplication in Stage 4.

In Stage 1, the client submits metachunk fingerprints F1, 12, A6 and 73 to the server. In Stage 2, the server eliminates metachunk fingerprints already saved, so it returns 12 and 73. In Stage 3, the data chunks and metadata associated with the "Missing" metachunks fingerprints are assembled and uploaded. In Stage 4, server-side deduplication saves any previously unsaved chunks.

3.3.1 Stage 1. Client: Chunk Processing and Query Construction.

The client performs several operations to construct the lookup query for the server, as shown in **SCAIL Algorithm, Stage 1** (see page 76).

SCAIL Algorithm, Stage 1: Build Chunks, Metachunks and Deduplication Query

/ Identical to* **Metadedup Write Algorithm** (p. 50)*, except for highlighted area. */*

1: **Client input:** target file name, client's master key key,
segment size S_{size}, chunk size C_{size}

2: Split file into chunks of average approximate size is C_{size}: $C = (c_1, c_2, \ldots, c_n)$

3: Perform Message Locked Encryption (MLE) on chunks C, generating
$CK = (ck_1, ck_2, \ldots, ck_n)$, the chunk encryption keys,
$EC = (ec_1, ec_2, \ldots, ec_n)$, the encrypted chunks, and
$CFP = (cfp_1, cfp_2, \ldots, cfp_n)$ are the chunk fingerprints,
where, for i in $[1, n]$:
$ck_i = \text{Hash}(c_i)$, is the chunk key,
$ec_i = \text{Encrypt}(c_i,\ ck_i)$, is the encrypted chunk, and
$cfp_i = \text{Hash}(ec_i)$ is the fingerprint of the encrypted chunk.

4: Gather chunk metadata $MD = (md_1, md_2, \ldots, md_n)$, where, for i in $[1, n]$:
$md_i = (cfp_i, len_i, k_i)$ is the chunk metadata,
cfp_i is the fingerprint (hash) of the encrypted chunk ec_i, and
len_i is $|c_i|$, the length of chunk c_i.

5: Calculate segment divisor: $D = \frac{S_{size}}{C_{size}}$

6: Group MD into metachunks $MC = (mc_1, mc_2, \ldots, mc_m)$, where, for j in $[1, m]$:
mc_j is a consecutive subsequence of MD entries, and
the sum of chunk lengths in mc_j is between ½ and twice S_{size}, and
only the final CFP in mc_j satisfies $(cfp \mod D) = 0$.

7: Extract plaintext chunk recipes from MC, generating
$PR = (pr_1, pr_2, \ldots, pr_m)$, the plaintext metachunk recipes,
where for j in $[1, m]$:
$pr_j = cfp$ from mc_j.

8: Perform Hybrid MLE on metachunks MC, along with PR, generating
$MK = (mk_1, mk_2, \ldots, mk_m)$, the metachunk encryption keys,
$PEMC = (pemc_1, pemc_2, \ldots, pemc_m)$, and
$MFP = (mfp_1, mfp_2, \ldots, mfp_m)$ are the metachunk fingerprints,
where, for j in $[1, m]$:
$mk_j = \text{Hash}(mc_j)$ is the metachunk key,
$emc_j = \text{Encrypt}(mc_j, mk_j)$
$pemc_j = (pr_j, emc_j)$, plaintext recipe and encrypted metachunk, and
$mfp_j = \text{Hash}(emc_j)$ is the metachunk fingerprint.

9: Store metachunk file recipe MFP, metachunk key recipe MK,
encrypted chunks EC, partially encrypted metachunks $PEMC$,
and the CFP recipes for those metachunks CFP_{mc}

10: Send metachunk file recipe MFP to the server

First, the file data is divided into chunks using a CDC algorithm. Each chunk undergoes MLE (or DupLESS), which includes generating a Chunk Fingerprint (CFP) by hashing the encrypted chunk.

The resulting stream of CFPs serves as input to another CDC algorithm. This algorithm determines segment boundaries based on the CFP values, as described in Metadedup (see Subsection 2.5.3 in the Related Works Chapter).

The encrypted metadata—including the fingerprint, encryption key, and chunk length—are collected to form a metachunk. After applying MLE to the metachunk, another cryptographic hash of the encryption produces the Metachunk Fingerprint (MFP).

Finally, the client creates File Recipes and Key Recipes using the list of MFPs and their corresponding encryption keys. The client then transmits lists of MFPs to the server to detect and eliminate previously saved metachunks and their constituent chunks.

3.3.2 Stage 2. Server: Metachunk Fingerprint Lookup.

The server executes a lookup operation to eliminate previously saved metachunks as shown in **SCAIL Algorithm, Stage 2** (see page 77). The MFP index is examined, and a list of "Missing" metachunk fingerprints representing segments this client has not stored previously is returned.

SCAIL Algorithm, Stage 2: Find Missing Metachunks

1: **Server input:** metachunk index MI
2: Receive: metachunk file recipe $MFP = (mfp_1, mfp_2, \ldots, mfp_m)$
3: Find missing metachunks $MM = (mm_1, mm_2, \ldots, mm_m)$, for j in $[1, m]$:
   If $mfp_j \notin MI$, add mfp_j to MM.
4: Send: missing metachunks MM to the client

3.3.3 Stage 3. Client: Chunk and Metadata Assembly and Upload.

The client assembles and uploads chunks and metadata as shown in **SCAIL Algorithm, Stage 3** (see page 79).

First, a "Missing" chunks recipe is constructed, which contains the CFPs associated with each MFP present in the list of "Missing" metachunks. The encrypted data for each chunk referenced in the "Missing" chunks recipe is then gathered for uploading.

To avoid redundancy, if a CFP appears multiple times, it is included only once in the upload. Additionally, the client generates a sorted list of CFPs for all the chunks in the upload.

Finally, the upload includes the encrypted metachunk data, the "Missing" chunks recipe, and the sorted list of CFPs.

SCAIL Algorithm, Stage 3: Assemble and Upload Missing Metachunks and Chunks

Client input: target file name, client key *key*
From Stage 1: $MFP, MC, PEMC, MK, EC$
Receive: Missing metachunk fingerprints $MM = (mm_1, mm_2, \ldots, mm_k)$
Initialise list of mfps and their cfps in recipe order: $UploadFP = ()$
Initialise upload metachunk/chunk data in UploadFP order: $UploadData = ()$
for each missing metachunk fingerprint *mfp* in MM **do**
    if $mfp \notin UploadFP$ **then**
        Find index j such that $MFP[j] = mfp$
        Retrieve partially encrypted metachunk $pemc = PEMC[j]$
        Append $(mfp, pemc)$ to $UploadData$
        Add *mfp* to $UploadFP$
        Retrieve metachunk $mc = MC[j]$
        Extract list of chunk metadata MD from mc
        for each chunk metadata $md = (cfp,\ len,\ ck)$ in MD **do**
            if $cfp \notin UploadFP$ **then**
                Find index i such that $CFP[i] = cfp$
                Retrieve encrypted chunk $ec = EC[i]$
                Append $(cfp,\ ec)$ to $UploadData$
                Add *cfp* to $UploadFP$
            end if
        end for
    end if
end for
Extract and sort CFPs from $UploadFP$ into $SortedCFP$
Encrypt metachunk key recipe: $EMK = \text{Encrypt}(MK,\ key)$
Upload to server: target file name, $MFP, EMK, UploadFP, UploadData, SortedCFP$

3.3.4 Stage 4. Server: Chunk Deduplication and Index Updates.

In Stage 4, the uploads of the multiple clients processed in Stage 1-3 are processed in a single batch. Their sorted CFP files are merged with all CFPs in the SCI bins (which contain all previously saved CFPs). In the client merged client stream, any repeated CFPs indicate cross-client duplicate chunks, as shown in Figure 1.3 and are added to the cross-client index. The merging between all client sorted CFPs and the previously saved CFPs from the SCI are performed simultaneously in a sequential pass through all the files. This process can be efficiently performed by holding the smallest CFP amongst all the clients and the SCI stream using a heapsort [84] data structure. This single pass through the files generates two indexes: the *previously-saved* and cross-client indexes.

In the SCI approach, unique vs. duplicate chunks are identified based on the sorted CFP order. However, chunks were allocated to containers while they were checked for being a duplicate, — in ascending chunk fingerprint order — this would result in the chunks of each file being fragmented across multiple containers, leading to inefficient restore operations. To address this issue, the SCAIL server performs chunk deduplication and updates the indexes in 2 parts. First, it creates an index of duplicate chunks using the SCI technique as shown in **SCAIL Algorithm, Stage 4, Part 1** (see page 81). The index is then used to identify and discard duplicates while allocating chunks to containers in recipe order in the following **SCAIL Algorithm, Stage 4, Part 2** (see page 82). It is important to note that most duplicates have previously been filtered out during the MFP-based client-side deduplication, requiring only a relatively small, memory-based duplicate index for each backup.

SCAIL Algorithm, Stage 4, Part 1: Find Duplicate and Cross-User Chunks

Server input: Sorted chunk index SCI with (fp, cid) mappings for stored chunks
Receive: multiple client $SortedCFP_{C1}, SortedCFP_{C2}, \ldots, SortedCFP_{Cn}$, from Stage 3
Sort-merge $SortedCFP_{C1}, SortedCFP_{C2}, \ldots, SortedCFP_{Cn}$ into $SortedCFP_C$
Initialise duplicates index DI, cross-user index CI
Initialise iterators: $i \leftarrow 0$ (for $SortedCFP_C$), $j \leftarrow 0$ (for SCI)
while $i < |SortedCFP_C|$ **and** $j < |SCI|$ **do**
    Find the fingerprint in the client stream: $cfp_c \leftarrow SortedCFP_C[i]$
    Find the stored fingerprint: $(cfp_s, cid) \leftarrow SCI[j]$
    if $cfp_c = cfp_s$ **then**
        Duplicate found. Update duplicates index: $DI[cfp_c] \leftarrow cid$
        Increment both i and j
    else if $cfp_c < cfp_s$ **then**
        Check if cfp_c occurs successively in $SortedCFP_C$
        if cfp_c appears multiple times in the client stream **then**
            Cross-user chunk found. Update cross-user index: $CI[cfp_c] \leftarrow nil$
        end if
        Increment i (move to the next client fingerprint)
    else
        Increment j (move to the next stored fingerprint)
    end if
end while
Store duplicate index DI, cross-user index CI

SCAIL Algorithm, Stage 4, Part 2: Allocate to Containers in Recipe Order

Server input:
Duplicate Index DI, Cross-user Index CI (from **SCAIL Stage 4, Part 1**, page 81)
On-disk sorted chunk index $SCI = ((fp, cid))$
Metachunk index MI, File Index FI
Memory cache persisting between calls $Cache = ((fp, cid))$
Metadata Containers, Data Containers
Receive: from Stage 3:
Target file name, $MFP, EMK, UploadFP, UploadData$
while *fp* in $UploadFP$ is a *mfp* **do**
    Retrieve (*mfp*, *pemc*) from $UploadData$
    Initialise empty list of data container ids $CIDSForSegment = ()$
    while next *fp* in $UploadFP$ is a *cfp* **do**
        Retrieve pair $(cfp,\ ec)$ from $UploadData$
        if $cfp \in DI$ **then**
            Skip this chunk (duplicate)
        else if $cfp \in CI$ **then**
            Retrieve *cid* from $CI[cfp]$
            if $cid \neq nil$ **then**
                Add *cid* to $CIDSForSegment$
            else
                Allocate *ec* to data container
                Add (cfp, cid) to *Cache*
                Add *cid* to $CIDSForSegment$
            end if
        else
            Allocate *ec* to data container
            Add (cfp, cid) to *Cache*
            Add *cid* to $CIDSForSegment$
        end if
    end while
    Store $(pemc, CIDSForSegment)$ into metadata container cid_{mc}
    Update metachunk index $MI[mfp] = cid_{mc}$
    if Cache is full **then**
        Sort *Cache* by fingerprint
        Flush sorted *Cache* to SCI
        Clear *Cache*
    end if
end while
Save MFP, EMK under target file name in file index FI

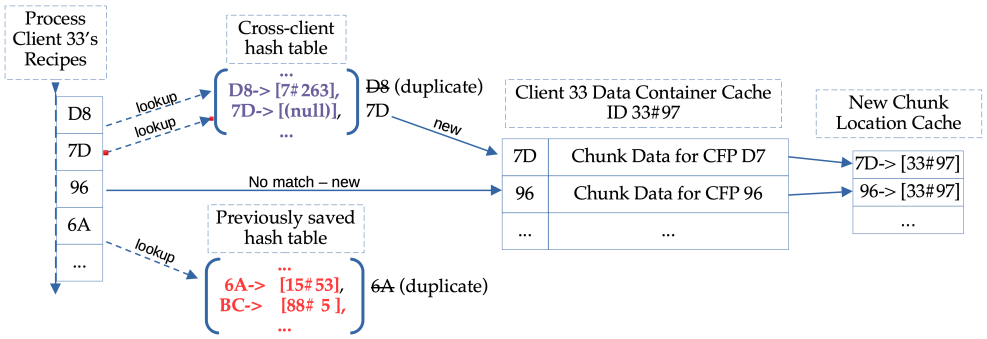


Figure 3.2: Example of Container Allocation in Stage 4. Starting on the left, CFPs, in recipe order are looked up in the cross-client and previously-saved hash tables. New chunks are allocated to the data container cache. The CFP and current container ID are then stored in the New Chunk Location Cache.

To allocate chunks to containers, CFPs in a client's recipe are looked up, in recipe order, in the cross-client and previously-saved hash tables (generated in Figure 1.3). If the chunk has been saved previously by any client (e.g., chunk 6A) or is a cross-client chunk allocated by some other client (e.g., chunk D8 by client 7), it is discarded. Otherwise, new cross-client chunks (e.g., chunk 7D), followed by new single-client chunks (e.g., chunk 96), are allocated to client 33's data container cache, which is flushed to disk when full. The CFP and container IDs are then saved in the New Chunk Location Cache. Not shown in the figure is the update of the cross-client hash table, replacing 7D's value of [null] with [33#97] once the container is flushed to disk.

The CIDs of any previously saved CFPs (extracted from the previously-saved index), along with the CIDs of any new chunks, are added to a list of CIDs for the segment. This list is saved in the manifest of the metadata container, associated with the segment MFP, and is used to find relevant data containers when fulfilling restore requests.

Finally, after allocating all data chunks to containers, the server updates the metachunk fingerprint index with the newly processed metachunks, indicating that the uploading client is now an owner. This last step enables subsequent lookups to detect that a specific client has stored a given metachunk and its data chunks.

3.3.5 SCAIL Restore

Performing a file restore in SCAIL requires a single query to the server. In **SCAIL Restore Operation** (see page 85), the client sends the file name to be restored to the server in line 2, and the server receives it on line 4 and obtains the associated list of metachunk fingerprints and their keys, which have been encrypted with the client's key.

SCAIL Restore Operation

1: **Client Input:** Target file name
2: Client sends target file name to the server

3: **Server Input:**
   File Index FI, Metachunk Index MI
   Metadata Containers, Data Containers
4: Receive: target file name from the client
5: Server retrieves MFP and EMK from FI
6: Filter stored metachunks and chunks, generating
   Encrypted metachunks $EMC_{file} = (emc_1, emc_2, \ldots, emc_m)$
   Encrypted chunks $EC_{file} = (ec_1, ec_2, \ldots, ec_n)$
   where, for j in $[1, m]$:
   Find the metadata container ID $cid_{metadata}$ at $MI[mfp_j]$
   Retrieve $(pemc, CIDSForSegment)$ from metadata container $cid_{metadata}$
   Extract (pr, emc) from $pemc$
   Add emc to EMC_{file}
   For each data container ID cid_{data} in $CIDSForSegment$:
      Load manifest for data container cid_{data}
      For each chunk fingerprint cfp_i in pr:
         If cfp_i is in manifest of cid_{data}:
            Retrieve ec_i from data container cid_{data}
            Add ec_i to EC_{file}
7: Server sends MFP, EMK, EMC_{file}, and EC_{file} to the client

8: **Client Input:** Client's master key key
9: Receive: MFP, EMK, EMC_{file}, and EC_{file} from the server
10: Decrypt metachunk key recipe: $MK = \text{Decrypt}(EMK, key)$
11: **for** each encrypted metachunk emc_j in EMC_{file} **do**
12: $mc_j = \text{Decrypt}(emc_j, mk_j)$
13: Extract chunk keys CK from mc_j
14: Decrypt chunks EC_{file} into C using CK
15: **end for**
16: Assemble chunks C to reconstruct the original file

In line 6, a list of encrypted metachunks (EMC) and encrypted chunks (EC) are built to return to the client. These are selected from the metachunks and chunks by processing one metachunk at a time. The metachunk is looked up in the metachunk index (MI), and the partially encrypted metachunk (PEMC), along with a list of container IDs for the chunks of the metachunk (CIDSForSegment), is retrieved from the

metadata container.

The encrypted portion of the PEMC is added to the list of EMCs to be returned to the client. The plaintext recipe (*pr*) of the PEMC contains chunk fingerprints for the segment. Each data container in CIDsForSegment is loaded, and if it contains a chunk fingerprint from *pr*, it is added to the list of encrypted chunks (ECs) to be returned to the client.

Finally, The server sends the MFPs, their encrypted keys (EMK) and the encrypted metachunks (EMCs) and encrypted chunks (ECs) to the client on lines 7.

On line 9, the client receives the data from the server, decrypts the metachunk key recipe on line 10, then processes each encrypted metachunk (EMC) on lines 11-15. Each EMC is decrypted with its key, revealing the keys to the chunks of the metachunk. The chunks are decrypted in line 14. Finally, in line 16, the chunks are assembled into the original file.

3.4 The Roles of Metachunks and Metachunk Fingerprints

The metachunk structure performs several roles in the design of SCAIL.

Grouping Chunks into Segments

Using CDC to form segments enables SCAIL to tolerate small changes from backup to backup and still have a reasonable probability of producing many of the same segments. Identical to Metadedup, we require a minimum segment size of one-half and a maximum segment size of double a specified target segment size (typically 2 MiB). Note that in SCAIL, a change to a single chunk in a segment will create a new metachunk, forcing all its chunks, even any previously stored chunks, to be uploaded. Alternatively, if all chunks in a segment are new, there is no excess upload. The first backup by a client is usually the largest single backup; for that client, all the chunks will be new additions to the server.

Duplicate-lookup Queries for Client-side Deduplication Requires No Disk I/O

Metadedup maintains both metachunk and chunk fingerprints in its RAM-based duplicate-lookup index, while SCAIL maintains only metachunk fingerprints. For 2 MiB segments from 8 KiB chunks, this reduces the memory requirements of the duplicate-lookup index 250 times, enabling the metachunk fingerprint index to identify massive amounts of previously backed-up physical data with only a small amount of memory. For instance, a 15 GB memory-based SCAIL metachunk fingerprint index with an average segment size of 2 MB and 30 bytes (B) for index elements (20 B SHA-1, 4 B container ID (CID), plus other metadata) could identify 1 PB of physical backup data. SCAIL would use an additional 2 GB of memory for disk-based server-side deduplication. A chunk-based index like Metadedup would be infeasible, requiring at least 3,750 GB of memory.

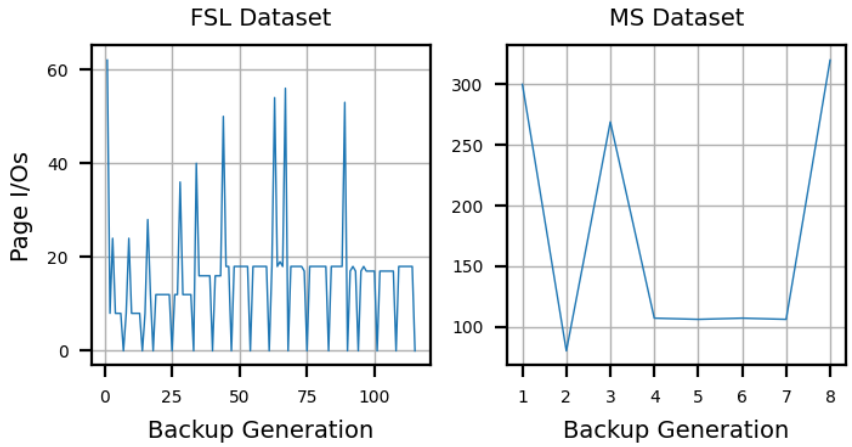
Metachunks Compress File Recipes

SCAIL's file recipes are lists of metachunk fingerprints rather than chunk fingerprints, producing file recipes up to 250 times smaller. Since metachunks are deduplicated as chunk data, the chunk list for a metachunk is stored only once.

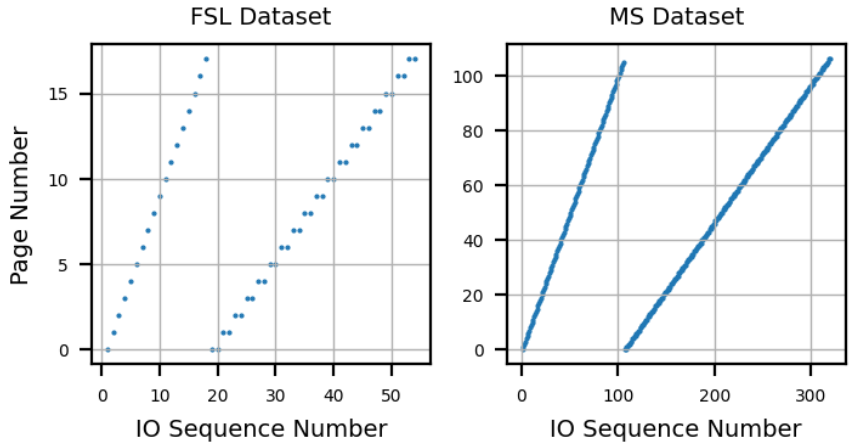
3.5 Server-side Chunk Deduplication

Sorted Deduplication uses a Log-Structured Merge-Tree (LSM tree) for its implementation of SCI. SCAIL will never perform an individual fingerprint query against the chunk index, so it uses an algorithm that takes advantage of this as follows:

New chunks are added to an in-memory cache containing a sorted list of $CFP \rightarrow CID$ mappings during deduplication on the server. When the cache becomes full, it is flushed to the disk. Figure 3.3 shows the disk I/O performed processing the FSL and MS datasets (for descriptions of the datasets, see Subsection 3.12.1). The top charts



(a) Number of page I/Os for each backup generation



(b) Page I/O detail for the 92nd (FSL) and 8th (MS) backup

Figure 3.3: Disk I/O the chunk-level deduplication in SCAIL Stage 4. The top charts show the number of I/O for each backup generation of the FSL and MS Datasets. The bottom charts show the I/Os for a single, selected backup generation.

(Figure 3.3a) show the number of page I/Os performed during the deduplication of chunks uploaded to the server. It can be seen that there are at most 350 I/Os per backup generation, and often is it significantly less than that. Usually, only a read pass through the sorted, disk-based index is required. However, if the cache reaches capacity, it must be flushed to disk. This appears as the periodic spike of increased I/O in these graphs.

The detailed I/O behaviour for the final cache flush for each dataset, which occurs on the 92nd backup for the FSL Dataset, and the 8th backup for the MS Data is shown in Figure 3.3b. The sequence of the I/Os is on the X-axis, and the page number accessed is on the Y-axis.

There are two passes through the pages, the first finding duplicates and the second flushing new fingerprints from the cache to each page. The flushing process first reads each page in the index in sorted order and determines whether adding the new fingerprints to the page's range of fingerprints would exceed the maximum page size. If not, the new fingerprints are appended to the file.

In the event of a split, two new pages are written, the first with the lower half and the second with the upper half of the consolidated fingerprints. Split pages start out at approximately half the maximum page size but are padded to the maximum page length (e.g. 128 MiB) to assist in keeping the later additions to the page on adjacent disk cylinders to improve write and read speed.

3.6 Implementing Ownership

SCAIL only needs to maintain ownership information on *metachunk* fingerprints, not on *chunk* fingerprints (as in Metadedup, see Section 2.5), since it only processes lookup or data restore requests at the metachunk fingerprint level.

3.7 Containers

SCAIL uses a *container system* to store encrypted chunks and metachunks. Separate container stores are used since these two data types have different average sizes and access patterns.

Data containers

To reduce the number of writes to disk when saving chunk data and to reduce the number of seeks when retrieving data, the server accumulates chunk data into a client-specific container, which is buffered in memory until filled, then written out. Containers have a *manifest header*, which maps fingerprints to data offsets in the container. Containers enable a small (e.g. 6-byte) CID to identify chunk storage locations.

Metachunk containers

A separate container space holds metachunks for each client. Additional information is stored in the manifest for each metachunk, specifically, a list of data CIDs containing chunk data for the segment the metachunk represents. This is used during restore requests.

SCAIL performs all lookup and restore requests at the metachunk level, so it does not disclose storage status at the chunk level. As with uploads, in the worst case, this can cause an entire segment of data to be downloaded to restore a single chunk of data.

SCAIL is *compatible* with mechanisms to support deletion operations. Reference counts can be employed in the manifest entries of data and metadata containers. A system of container reference indirection, as described in Sorted Deduplication [36], can be used to consolidate under-utilised containers.

3.8 Redundant Data Uploads

We define redundant uploads as any metachunk or chunk data uploaded and found to have been saved previously on the server. Note that SCAIL's server-side deduplication removes all redundantly uploaded data, preserving exact deduplication. These excess uploads can have several causes:

- **Cross-client redundancy in backup uploads.** This occurs when two or more

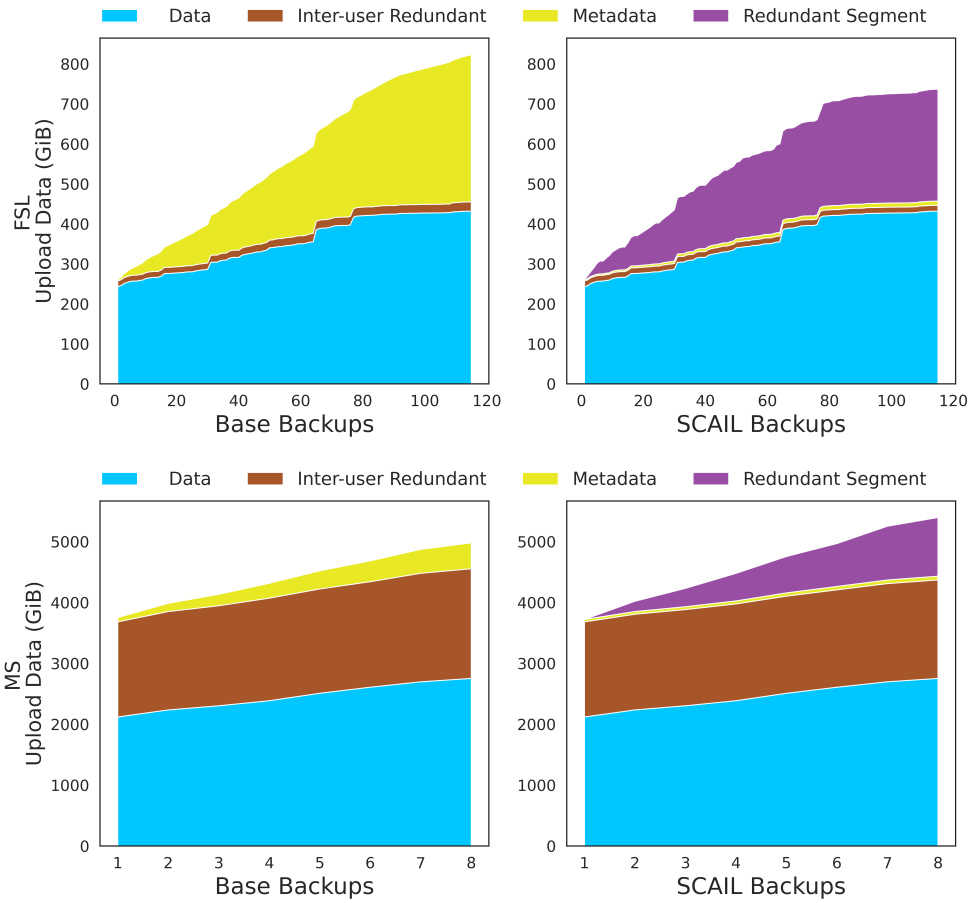


Figure 3.4: Breakdown of the cumulative upload volume by component type for the FSL (top) and MS Dataset (bottom), comparing the Base and SCAIL techniques. SCAIL substantially reduces metadata upload volume, but introduces RSD upload volume.

clients upload new data in the same backup generation processed by the server. Each client would have made a lookup request to the server and been notified that the metachunk had not been stored previously. These redundant uploads are typical on the first backup of groups of devices in corporate environments where many clients have the same operating system and application software. In the MS dataset (described in Subsection 3.12.1) in Figure 3.4, the first backup of 140 client volumes has 2.1 TiB of data and 1.5 TiB of cross-client redundant data upload. This produces a cross-client excess upload volume of 42% for Base *and* SCAIL. Subsequent clients uploading the same data would experience cross-client backup redundancy, as described next.

- **Cross-client backup redundancy for data privacy.** When a client performs a lookup, another client may have already uploaded the metachunk on a previous backup. But to avoid side-channel leakage, the system must deny existence for non-owners, forcing a redundant upload. After these redundant uploads have been processed, the client is registered as an owner of the data. In subsequent backups, these cross-client redundant uploads on the same data will no longer occur.

- **Redundant Segment Data (RSD).** This type of redundant upload is caused by SCAIL's technique of coarse client-side deduplication. It only occurs when chunks making up the segment of a 'new' metachunk have already been backed up as part of the chunks of some other previously uploaded segment. A worst-case example would be a single chunk modified from a previously stored metachunk. A 'new' metachunk would be generated, all chunks of its segment uploaded, and all but one of its chunks would have been uploaded previously.

RSD uploads could be avoided if the client waited for a second response from the server's chunk-level index or if the client maintained and consulted a list of chunks they had uploaded previously. However, we found that the additional uploads had a

surprisingly small cost factor and impact on system performance. As best we could determine, the primary cloud hosting services do not charge for uploads (e.g., [7, 66]), and most consumer broadband plans do not charge by upload volume.

As seen in Figure 3.4, no RSD is uploaded in the first backup since all the metachunks are new. Assuming a 60Mbps upload speed, the average FSL client for both SCAIL and Base (with no metadata deduplication) would take 77 minutes to upload their first backup of 32.2 GiB. Each of the subsequent 114 uploads would take an average of 89 seconds of upload time for each of the eight clients in Base (637 MiB) and 32 seconds for SCAIL (232 MiB). For the MS dataset, the first upload would take 63.5 minutes (26.5 GiB), and the upload time per backup for one of the (on average) 92 devices would take 3 minutes (1.1 GiB) for Base and 4 minutes (1.5 GiB) for SCAIL.

The additional uploaded data must be stored on the server until it is processed, and then it will be discarded. As noted in Sorted Deduplication [36], the performance of a SCI deduplication system is primarily driven by the size of the previously backed-up data rather than the size of the input data, so the additional RSD has a minimal impact on processing time.

We present the threat model for SCAIL and introduce a security analysis of its design. We show that SCAIL maintains data privacy, even in the presence of internal and external attackers. Our analysis considers practical attack scenarios and evaluates the computational feasibility of potential attacks.

3.9 Threat Model

In this section, we define the threat model for SCAIL, classifying adversaries into two categories: internal and external attackers. This analysis specifically addresses SCAIL's unique architecture and capabilities and includes key differences from the threat model of Metadedup, as described in Section 2.5.7.

3.9.1 Internal Attackers

Internal attackers such as hackers, malicious system administrators, or cloud service provider employees have gained access to the backup server. In the context of SCAIL, such attackers are assumed to have access to the following:

- Encrypted data chunks and their lengths.
- Chunk fingerprints (CFPs).
- Encrypted metachunks and their lengths.
- Metachunk fingerprints (MFPs).
- The metachunk fingerprint index.
- The SCI (Sorted Chunk Index).
- Plaintext sorted lists of CFPs within metachunks.

3.9.2 External Attackers

External attackers attempt to obtain data they are not authorised to access without having direct access to the server. These adversaries may try to exploit the deduplication process to infer information about stored data. We assume that communication channels between the client and server are secure using protocols such as SSL/TLS. We focus primarily on **side-channel attacks**, as they present the most significant threat by an external attacker to data confidentiality.

3.10 Security Analysis

3.10.1 Data Confidentiality

Chunk Encryption

As mentioned above in section 2.2.1, data chunks in SCAIL can be encrypted using either MLE [8] or DupLESS [38]. MLE derives encryption keys directly from the content of the chunks. However, MLE does not provide data privacy for predictable chunk

sets—chunks that are widely known, easily guessed, or structured in a predictable pattern (e.g., standard file headers, known template patterns, or common files).

DupLESS provides data security even for predictable chunks by generating the encryption key k through a key server that relies on a global secret s. The client sends the chunk fingerprint $\text{Hash}(c)$ to the server, which computes $k = \text{F}(\text{Hash}(c), s)$, where F is an Oblivous Psuedo Random Function. This ensures that the key k cannot be derived from the fingerprint $\text{Hash}(c)$ alone, as it requires access to the global secret s.

Metachunk Construction and Encryption

A metachunk records the list of CFPs that constitute a segment, along with the lengths of the chunks and the encryption keys for each chunk. In SCAIL, when a metachunk is encrypted, it is split into 2 parts. The first part is a sorted, plaintext list of the constituent CFPs for the metachunk. This sorted list is accessible to internal attackers. The second part consists of an offset list into the plaintext chunks, (which can be used to restore the CFP order), the length of each chunk and the encryption key of each chunk. The second part is encrypted using MLE. The metachunk encryption key is encrypted with a client-specific key, so that only the client can decrypt the metachunk.

By separating the plaintext sorted list of CFPs from the encrypted offsets, lengths and keys, SCAIL avoids an additional server roundtrip during restore operations while maintaining security.

3.10.2 Internal Attack Scenarios

An internal attacker with access to the stored data attempts to reconstruct the original data segments by:

1. Accessing the plaintext sorted list of CFPs from a metachunk.
2. Attempting to determine the lengths and encryption keys for each chunk corresponding to the CFPs.

3. Reconstructing the original ordering of chunks by guessing the sequence represented by the encrypted offsets.
4. Encrypting the metachunk (using guessed chunk encryption keys and sequence) and comparing the resulting MFP to the one stored on the server to validate their guess.

We analyse the computational effort required for this attack and demonstrate its infeasibility.

Step 1: Determining Chunk Lengths and Encryption Keys

Since the internal attacker has access to the CFPs, their task is to find the corresponding chunk lengths and encryption keys. The length can be determined by searching the SCI index for the target chunk, loading the data container where it is stored, and noting the encrypted chunk length. The number of possible encryption keys depends on the chunk content.

For each CFP, the attacker could attempt to generate possible chunks and derive the encryption keys. However, unless the chunk content is highly predictable, the number of possible chunks for each CFP is vast. Additionally, if DupLESS is used for chunk encryption, the attacker cannot derive the encryption keys without compromising the key server's global secret.

Step 2: Reconstructing the Original Ordering

The attacker knows the sorted list of CFPs but not the original sequence of chunks in the segment. The encrypted offsets within the metachunk represent a permutation of the sorted list.

The total number of possible permutations of the set of chunks would be $n!$. For segments of average size 2MiB, and chunks averaging 8KiB, the minimum number of chunks in metachunk would be 128. So the possible permutations would be 128!.

Step 3: Encrypting Metachunk Candidates

For each possible permutation of the chunk sequence, the attacker would need to:

1. Arrange the chunks according to the permutation.

2. Encrypt the candidate chunks.
3. Assemble the encrypted chunks' metadata into a metachunk.
4. Encrypt the metachunk using MLE.
5. Compute the MFP by hashing the encrypted metachunk.

Caching Optimisation

The attacker can cache the encrypted chunks and their metadata after determining them in Step 1, avoiding re-encrypting chunks for each permutation. However, the need to consider all permutations remains, as the order of chunks affects the encrypted offsets and, consequently, the encrypted metachunk and its fingerprint.

Total Computational Effort

Even with the cached encrypted chunks, the attacker must encrypt $n!$ metachunk candidates and compute each of their fingerprints.

Assuming that the time to encrypt a metachunk and compute its fingerprint is $T_{\text{metachunk}}$, the total time required is:

$$T_{\text{attack}} = n! \times T_{\text{metachunk}}.$$

For $n = 128$, $n!$ is approximately 3.86×10^{215}. Even if $T_{\text{metachunk}}$ is extremely small (e.g., $1\,\mu$s), the total time is:

$$T_{\text{attack}} = 3.86 \times 10^{215} \times 1\,\mu\text{s} = 3.86 \times 10^{209}\,\text{seconds},$$

or 1.22×10^{202} years, which is impractical.

Handling Small Metachunks

Metachunks generated from the metadata of only a few chunks and encrypted with MLE will be much more susceptible to brute-force attacks. The number of permutations an attacker would need to generate is expressed by $N = \frac{n!}{(n-c)!}$, where c is the

number of chunks in a segment (see Section 2.5.7). If c is small, the denominator approaches n, making the number of permutations feasible for brute-force attacks.

To mitigate this vulnerability, SCAIL processes incoming files as continuous streams of bytes and pads the final generated metachunk if necessary. Small files are combined with subsequent data to form larger metachunks, maintaining a high combinatorial complexity.

Impact of Predictable Chunks

If the chunk content is highly predictable, the attacker may have an easier time determining the chunk data and encryption keys. However, when using DupLESS for chunk encryption, the attacker cannot derive the encryption keys without access to the key server, even if the chunk content is known.

In the worst-case scenario where the attacker knows all chunk data and encryption keys, they still face the challenge of permutation. The total number of permutations remains $n!$, preserving the computational infeasibility of the attack.

Security of the Plaintext CFP List

While the plaintext sorted list of CFPs is accessible to internal attackers, it does not reveal the original sequence of chunks in the segment. Without the encrypted offsets, which are protected by the metachunk encryption, the attacker cannot reconstruct the original data order.

Moreover, since clients request metachunks by their MFPs and receive the encrypted metachunk along with the encrypted chunks, they are never allowed to query the server for the storage status of individual chunks or attempt to restore individual chunks. This design reduces the risk of side-channel attacks that could exploit chunk access patterns (see Side-Channel Attacks below).

Effect of Key Server Compromise

If the key server used in DupLESS is compromised, the security of chunk encryption reverts to that of standard MLE. While this reduces protection against brute-force attacks on predictable chunk sets, the overall security of SCAIL remains robust due to:

- The high computational cost of permutation attacks on metachunks.
- The secure handling of encryption keys for metachunks, which are protected by client-specific keys.
- The measures taken to handle small files and maintain large metachunk sizes.

3.10.3 External Attack Scenarios

External attackers attempt to obtain data they are not authorised to access without having direct access to the server. These adversaries may try to exploit the deduplication process to infer information about stored data. We assume that communication channels between the client and server are secure using protocols such as SSL/TLS. We focus primarily on **side-channel attacks**, as they present the most significant threat by an external attacker to data confidentiality.

Side-Channel Attacks in Deduplication Systems

Side-channel attacks exploit information leakage when a server reveals whether a particular piece of data already exists in storage. An adversary can utilise this information to confirm the presence of specific files or infer sensitive information.

Common Side-Channel Attack Scenarios

Confirmation-of-File Attack

An adversary attempts to upload a target file and observes the server's response. If the server indicates that the file is a duplicate and does not require uploading, the adversary can deduce that the file already exists on the server, implying that another user has uploaded it [34].

Content Guessing Attack

By systematically uploading files with known content and monitoring deduplication responses, an adversary can ascertain whether certain data segments are stored on the server, potentially revealing confidential information [34].

Learning-the-Remaining-Information (LRI) Attack

An adversary who possesses most of a file's content but lacks some sensitive parts can attempt to guess the missing pieces. By uploading files with different variations of the unknown content and analysing deduplication results, the adversary aims to learn the missing information [32].

Mitigations Implemented in SCAIL

SCAIL incorporates several design choices to mitigate side-channel attacks:

1. **Client-Side Deduplication Limited to Own Data**

   Clients perform deduplication only against data they have previously uploaded. Consequently, when a client uploads data, it cannot determine whether other clients have uploaded the same data.

   Mitigation: Prevents adversaries from using deduplication responses to infer the presence of other users' data.

2. **Server-Side Deduplication Without Disclosure**

   The server performs cross-client deduplication but does not reveal deduplication status to clients. Clients are always instructed to proceed with the upload process as if the data were new, regardless of whether it already exists on the server.

   Mitigation: Ensures that adversaries cannot gain information about stored data based on the server's responses.

3. **Uniform Server Responses**

   The server will disclose that a metachunk has been previously uploaded, *if and only if that client has previously uploaded the metachunk.*

Mitigation: Eliminates differences in communication patterns that could be exploited in side-channel attacks.

4. **Data Upload Policies**

   Clients are required to upload data chunks or metachunks regardless of whether other clients have uploaded them.

   Mitigation: Prevents adversaries from avoiding data uploads to confirm the existence of data.

Analysis of Side-Channel Attack Resistance

We analyse how the design of SCAIL resists common side-channel attacks.

Confirmation-of-File and Content Guessing Attacks In SCAIL, since the server's responses do not reveal whether data has already been uploaded by some other client, adversaries cannot confirm the presence of specific files or data chunks that they have not uploaded.

Learning-the-Remaining-Information (LRI) Attacks Adversaries cannot leverage deduplication responses to learn unknown parts of a file because the server does not disclose the deduplication status for other clients. The requirement for clients to upload all data, even if it has been uploaded by some other client, combined with the inability to detect whether cross-client deduplication occurs, prevents attempts to learn missing information.

Impact on Legitimate Clients These mitigations help maintain client data privacy, but require the client to upload data that is already on the server. We feel this trade-off is acceptable since it maintains the data confidentiality and prevents side-channel leakage.

3.10.4 Summary

Our analysis demonstrates that SCAIL effectively maintains data confidentiality and integrity, even in the presence of internal and external attackers. By encrypting data chunks with DupLESS or MLE, securely managing encryption keys, and implementing mechanisms for metachunk construction and encryption, SCAIL mitigates the risks posed by potential attacks.

The inclusion of a plaintext sorted list of CFPs does not compromise security, as attackers still face a prohibitively large number of permutations when attempting to reconstruct the original data. By ensuring that metachunks are of adequate size by merging small files in a streaming manner, SCAIL maintains a large combinatorial complexity, making brute-force attacks computationally infeasible.

Integrity and Availability

Like Metadedup, SCAIL can be integrated with existing integrity verification schemes, such as data auditing protocols [6, 21]. These mechanisms protect against malicious modifications or deletions by the server, ensuring its integrity.

Availability and resilience for SCAIL can be enhanced through deduplication-aware secret sharing with CDStore [46]. Building SCAIL upon CDStore leverages convergent dispersal and multi-cloud storage to improve data resilience and maintain high availability. By splaying data across multiple independent servers using CDStore's secret-sharing mechanisms, SCAIL can ensure that the loss or failure of any single server does not compromise data availability. An additional benefit is that the security model would require an attacker to breach multiple servers to gain unauthorised access to the data, significantly increasing the difficulty of potential attacks.

3.11 Limitations

Next, we examine the limitations of SCAIL, highlighting challenges when dealing with massive datasets, relatively small datasets, resource contention, minimally evolving datasets and read/write amplification. We also note the computational overhead of encryption, client-side deduplication restrictions, dependence upon batch-oriented processing and synchronisation requirements among multiple clients.

3.11.1 Scalability Constraints with Very Large Datasets

The performance of SCAIL when handling massive datasets—specifically those containing more than a few petabytes of unique data—is uncertain. The system relies on in-memory data structures for mapping MFPs to the on-disk storage of their associated metachunk. With RAM constrained to 256 GiB, scaling these structures to accommodate multi-petabyte datasets becomes challenging. The limited RAM may lead to increased disk I/O operations, adversely affecting performance and throughput.

3.11.2 Performance Compared to RAM Index-Based Systems

For datasets with unique data smaller than 100 TB, systems utilising RAM-based indexes or other low-I/O deduplication mechanisms may outperform SCAIL. Such systems can hold the entire deduplication index in memory, enabling faster lookup and deduplication operations. In contrast, our system may experience higher latency due to its use of the disk-based SCI technique for chunk-level, cross-user deduplication.

3.11.3 Impact of Limited Client Numbers on Resource Contention

The advantages of SCI, particularly in eliminating resource contention, diminish in environments with a limited number of clients. SCI is designed to optimise deduplication processes across numerous clients by organising and merging data efficiently with very little disk I/O requirements. When the client base is small, the overhead

associated with merge-sorting may not be justified, rendering simpler deduplication strategies more effective.

3.11.4 Challenges with Low-change Datasets

SCAIL is at a disadvantage in processing datasets that don't evolve much over time. In scenarios where datasets exhibit very small deltas between versions, traditional disk-based chunk-level indexing systems may process data faster than our system. The need for a full SCI chunk index pass in our approach can introduce unnecessary overhead, especially when only minor changes occur between backups.

3.11.5 Read and Write Amplification Issues

SCAIL exhibits read and write amplification, affecting bandwidth usage and performance.

Read Amplification SCAIL is subject to reduced restore performance due to chunk fragmentation issues inherent in container-based deduplication systems, as described in [48]. These issues lead to read amplification, where restoring data requires reading more than the necessary chunks.

SCAIL is compatible with schemes designed to mitigate the reduction in restore performance. However, SCAIL introduces an additional source of read amplification. Restoring a small portion of data will necessitate downloading an entire segment, as data is retrieved at the segment level. This inefficiency arises from the transfer of unused data during partial restores.

Write Amplification A single changed chunk in a segment requires uploading the entire segment. This results in increased storage and bandwidth consumption, as more data than necessary is transferred and temporarily stored.

3.11.6 Limitations Due to Client-Side Deduplication Restrictions

To protect against side-channel attacks and ensure data privacy, SCAIL avoids cross-client deduplication on the client side. While this approach enhances security by preventing potential leakage of information through deduplication patterns, it also means that clients cannot benefit from the deduplication of data that exists on other clients. Consequently, bandwidth usage will increase. Fortunately, storage requirements are not increased since SCAIL performs exact, chunk-level deduplication when storing data on the server.

3.11.7 Computational Overhead from Encryption

Providing data privacy guarantees necessitates the encryption of data both in transit and at rest. The encryption and decryption processes introduce computational overhead on both client and server machines. This overhead can impact the overall performance of backup and restore operations, particularly in environments with limited computational resources or where performance is a critical concern.

3.11.8 Dependence on Batch Uploads

The system's requirement to wait for batches of client data before initiating server-side deduplication can lead to delays in data processing. This limitation is especially pertinent in environments where immediate data backup is necessary. The reliance on batch uploads contrasts with systems that support continuous or real-time deduplication, potentially reducing the system's suitability for certain applications.

3.11.9 Single Batch Server-side Deduplication Limitation

We analysed only a single batch cross-user, server-side deduplication pass at a time. This design choice makes the server wait for all client data to be uploaded before initiating deduplication on *any* client. In contrast, Sorted Deduplication proposed running

multiple merge and store operations concurrently on a single datastore. The inability to process multiple deduplication passes simultaneously may limit the system's throughput and responsiveness in high-load or real-time environments.

3.12 Evaluation

We wrote Python implementations of Base (traditional chunk-based deduplication, no segment groupings, no metadata deduplication), Metadedup and SCAIL encrypted deduplication systems and conducted experiments against trace-driven simulations of different backup workloads.

3.12.1 Trace-driven Simulation

We evaluate SCAIL via trace-driven simulation.

Experimental Datasets

We use two real-world publicly available datasets:

- **FSL** The FSL dataset originates from the File systems and Storage Lab (FSL) at Stony Brook University, as documented in [79]. It is the most commonly used dataset in the literature, having been used in over 50 publications, including over a dozen since 2020 [70, 20, 85, 50, 40, 78, 99, 86, 102, 103, 62, 47, 52, 42, 101]. We replicated the subset identified in Metadedup from the *fslhomes* 8 KiB traces. These traces encompass periodic captures of home directory contents belonging to eight students on a communal network file system. The original FSL snapshots provide variable-size chunk fingerprints, which were 48-bit in format, which we have hashed to produce 20-byte SHA-1 values and have retained their associated metadata details (i.e. file length, inode, chunk length). This included all snapshots from January 22 to June 17, 2013, which were aggregated daily and

produced 115 backups. These backups comprised 56.2 TiB of logical data and 431.9 GiB of physical data.

- **MS** This public dataset was collected from desktop computers at Microsoft [58]. In addition to Sorted Deduplication, it has been used in several recent papers where large numbers of clients and/or high data volume need to be simulated [40, 62, 42, 63, 27]. It consists of 857 Windows file system snapshots. Like Sorted Deduplication, we collected the backup histories for the 140 distinct client streams. Their selection was random, but we desired a reproducible dataset, so we selected the 140 with the earliest backup time. To simulate large numbers of concurrent client backups, we grouped them into eight weekly backup generations. Not all systems participated or were collected in every weekly backup. The maximum number of client streams was 140, the minimum was 64, and the average was 102 devices per backup generation. The chunks used were generated using CDC with a target chunk size of 8KiB [69]. This short-term dataset of backups for 140 devices occupies 45.6 TiB of logical data and 2.7 TiB of physical data.

Methodology

Our simulator allows us to vary segment sizes and accept different backup workloads. The traces representing client files for the FSL or MS backups are read from backup folders in creation order, and the simulator runs through the four steps (see Subsection 3.3) of encrypted deduplication and we gather metrics.

3.13 Evaluation Results

We performed backup operations on the FSL and MS datasets with 512 KiB, 1 MiB, 2 MiB and 4 MiB segment sizes. For our SCI implementation, we used 128 MiB pages, the same as Sorted Deduplication. We also used 128 MiB containers.

Table 3.1: Segment Size Effect on Memory and Upload Size in SCAIL

	Components/Metrics	512KiB	1MiB	2MiB	4MiB
FSL	**Index Memory**				
	Base (GiB)	1.400			
	SCAIL Intra-user (GiB)	0.032	0.018	0.010	0.006
	SCAIL Inter-user (GiB)	0.250	0.250	0.250	0.250
	Total SCAIL (GiB)	**0.282**	**0.268**	**0.260**	**0.256**
	Memory Change	**-79.9%**	**-80.9%**	**-81.4%**	**-81.7%**
	Upload Volume				
	Base (GiB)	825			
	SCAIL (GiB)	628	676	736	805
	Upload Change	**-23.9%**	**-18.1%**	**-10.8%**	**-2.5%**
MS	**Index Memory**				
	Base (GiB)	10.358			
	SCAIL Intra-user (GiB)	0.197	0.110	0.062	0.035
	SCAIL Inter-user (GiB)	2.0	2.0	2.0	2.0
	Total SCAIL (GiB)	**2.197**	**2.110**	**2.062**	**2.035**
	Memory Change	**-78.8%**	**-79.6%**	**-80.1%**	**-80.4%**
	Upload Volume				
	Base (GiB)	4,992			
	SCAIL (GiB)	5,019	5,177	5,397	5,679
	Upload Change	**0.5%**	**3.7%**	**8.1%**	**13.9%**

Memory Use

For step 2 and step 4, we used memory-based indexes for Base and Metadedup. The Base index contained chunk fingerprints only, and Metadedup's index included metachunk fingerprints as well, so was 1% larger. SCAIL used a metachunk fingerprint-only memory-based index for step 2. For step 4, SCAIL used 128 MiB of memory for loading a single chunk index page in each dataset. SCAIL also allocated a write cache for the SCI, 128 MiB for the FSL dataset, and 1.875 GiB for the MS dataset.

Table 3.1 presents the change in memory requirements and the difference in upload volume produced after storing all backups. The figures shown are the memory and disk-volume measurements taken after running all client backups for the FSL (upper group) and MS (lower group) Datasets.

The first line in each group shows the volume of memory consumed by the CFP

index used by the Base scheme. The "SCAIL Intra-user" measurements reflect the volume of memory consumed by SCAIL's MFP-only index, and the "SCAIL Inter-user" measurement reflects the volume SCAIL requires by using SCI to perform chunk-level deduplication on the server.

Upload volumes shown at the bottom of each group differ between Base and SCAIL because SCAIL deduplicates metadata, so it does not have to upload repeated File/Key Recipes. These savings are offset required by RSD uploads in SCAIL, which increase as segment size increases.

As segments increase in size, there are fewer of them to store, decreasing the required memory size. SCAIL uses 79.9-81.7% less memory than Base for the FSL dataset, and 78.8-80.4% less memory for the MS dataset.

Chunk Index Disk I/O

The left side of Figure 3.3a shows SCAIL's chunk index disk I/O for the FSL dataset. Over 115 backup generations, the deduplication process stored 431 GiB of unique chunk data with an effective chunk size of 7.7 KiB. The chunk index must store 58.5 million chunks; at 30B per chunk, it would occupy 1.6 GiB of disk space. SCAIL ended up with eighteen pages and reads an average of 93 MiB per page consisting of 3.2 million fingerprints. The spikes on the chart are the system flushing the new fingerprint cache if it becomes full.

Upload Overhead

SCAIL's use of coarse client-side deduplication caused it to upload redundant chunk data (see Subsection 3.8). For the FSL dataset, though, the 97% reduction in metadata upload more than offset additional uploads of redundant chunk data; see Table 3.1. The net effect was to reduce upload size compared to Base from 2.5-23.9%. For the MS dataset, which only had eight backups and only a 86-89% reduction in metadata

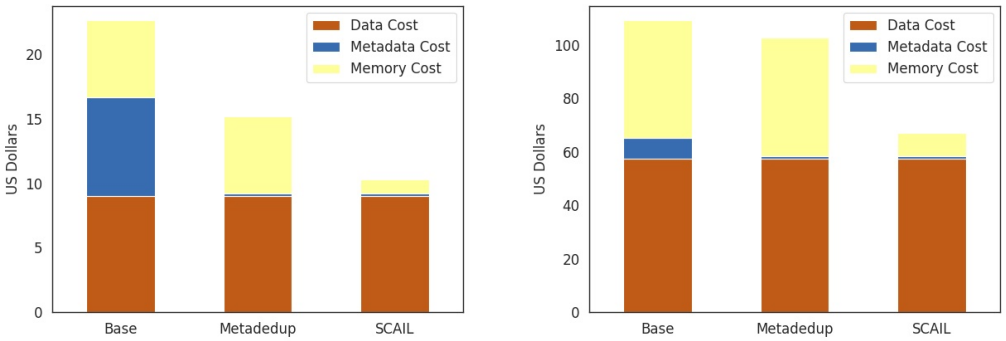


Figure 3.5: Stacked bar chart of total costs after all backups, showing the breakdown of deduplicated data, metadata and memory costs for the FSL (top) and MS (bottom) datasets.

upload, SCAIL increased upload size compared to Base by 0.5-13.9%.

Memory and Storage Costs

We evaluate the memory and storage costs by comparing Base, Metadedup and SCAIL. A chart comparing the costs for the FSL and MS datasets is shown in Figure 3.5. All schemes use an average of 8 KiB-sized chunks. We use 2 MiB segments since Metadedup [44] found this provided the best metadata savings for the FSL dataset. We surveyed memory and storage costs from www.amazon.com in February 2022 and used US dollars of $4.25/GiB for memory ($68 for Crucial 16 GiB Single DDR - CT16G4SFD824A) and $21.30/TiB for storage ($310 for Seagate 16 TB HDD Exos - ST16000NM001G).

For the FSL dataset, Metadedup and SCAIL reduce Base's metadata storage requirements from 369.6 GiB to 10.4 GiB (97.5%) and 10.2 GiB (97.2%), respectively. This reduces metadata storage costs from Base's $7.69 to 22 cents for Metadedup and 25 cents for SCAIL. Physical data storage for each scheme is 431.9 GiB, costing $8.99. Metadedup has a slight increase over Base's memory size of 1.385 GiB to 1.393 GiB, while SCAIL slashes memory to 260 MiB. This resulted in memory costs of $5.89, $5.92 and $1.10 for Base, Metadedup and SCAIL, respectively. Base's total memory and stor-

age costs are $22.56, for Metadedup are $15.12, and for SCAIL are $10.34. The savings over the price for Base are 33.2% for Metadedup, and for SCAIL they are 54.3%, which is 21.1% additional savings.

For the MS dataset, Metadedup and SCAIL reduce Base's metadata storage from 378.1 GiB to 50.1 GiB (89.2%) and 50.4 GiB (86.3%), respectively. This dataset had a large volume of new data over a relatively short number of backups, which provided fewer opportunities to reduce metadata storage. Nonetheless, Metadedup's metadata deduplication techniques reduced metadata storage cost from $7.87 for Base to $1.04 for Metadedup, and $1.05 for SCAIL. The large volume of physical data storage (2,753.7 GiB) for each scheme cost $57.29 to store. Required memory went from 10.3 GiB for Base to 10.4 GiB for Metadedup and was reduced to 2.1 GiB for SCAIL. This resulted in memory costs of $44.02, $44.28, and $8.76 for Base, Metadedup and SCAIL respectively. So total memory and storage costs ended up being $109.18 for Base, $102.62 for Metadedup and for for $67.10 SCAIL. The cost savings for Metadedup over Base are 6.0%, and for SCAIL, it is 38.5%, which is 32.5% additional savings.

3.14 Summary

We present SCAIL, which builds upon the metadata storage savings introduced by Metadedup and integrates it with the low-memory requirements and high client capacity of Sorted Deduplication. SCAIL performs coarse-grained client-side deduplication, which causes the upload of previously saved chunks. Compared to traditional encrypted deduplication systems, SCAIL significantly reduces metadata storage, required memory capacity and index disk I/Os while providing confidentiality guarantees for both data and metadata.

Chapter 4

P-SCAIL: Parallel SCAIL

4.1 Introduction

We now turn our focus to increasing server throughput. Higher throughput enables a system implementing P-SCAIL to fully utilise the increased data storage and higher concurrent client capacities introduced by SCAIL. In P-SCAIL we:

- We reduce SCI cache requirements compared to SCAIL with an improved cache flushing technique.
- We utilise data and task parallelism to take advantage of multiprocessor servers, achieving significant speedups in client-side and server-side deduplication throughput.

A quick response time to lookup queries (see Stage 2 in Figure 3.1) is critical for any client-side deduplication system, since clients cannot assemble and upload data until they have received a deduplication response from the server. SCAIL's response to lookup queries is very fast, but P-SCAIL further reduces client response times by processing multiple client queries in parallel. P-SCAIL also parallelises server-side deduplication to increase metadata processing throughput, which increases the number of concurrent clients that a server can accommodate in a given timeframe.

In the following sections, we first outline P-SCAIL's parallel strategies for client-side deduplication, and then delve into the use of data and task parallelism to enhance

the simultaneous multiclient, batched deduplication method employed for server-side deduplication.

4.2 Parallel Client-side Deduplication

Clients send MFPs as lookup queries to the server. A separate process from a process pool handles each client request in turn to enable parallel processing, so it is trivially parallel. The server will return the set of "Missing" MFPs that have not been saved by that client previously.

4.3 Batched, Parallel Server-side Deduplication

SCAIL employs a batched approach to concurrently handle the upload data from multiple clients. The system efficiently processes these data batches by leveraging the sequential access pattern of SCI. P-SCAIL enhances SCAIL's batch approach and uses both data and task parallelism to reduce batch processing times.

Inspired by Sorted Deduplication [36], we perform data parallelism by assigning ranges of CFPs to specific processors and perform SCI in parallel. A given processor manages all SCI operations within its specified range of CFPs.

Although it limits us to 256 processes and the number of processes must be divided into 256 evenly, for efficiency, processors work on a range of fingerprints based on the most significant byte of the CFP. For n processors, the i-th processor (where $i = 0, 1, \ldots, n-1$) will handle fingerprints falling within the range:

$$\left(\frac{i \cdot 256}{n}\right) \text{ to } \left(\frac{(i+1) \cdot 256}{n}\right) - 1,$$

For example, with $n = 8$, the 0-th processor will handle chunks whose most significant

byte has a lower and upper bound of:

$$\text{Lower bound} = \frac{0 \cdot 256}{8} = \frac{0}{8} = 0, and$$

$$\text{Upper bound} = \frac{(0+1) \cdot 256}{8} - 1 = \frac{1 \cdot 256}{8} - 1 = \frac{256}{8} - 1 = 32 - 1 = 31.$$

Since the total number of possible byte values is 256 (ranging from 0 to 255), dividing this range evenly among 8 processors results in each processor handling $\frac{256}{8} = 32$ byte values.

Before uploading, clients divide their sorted chunk fingerprint files into separate files based on the number of processes.

Each process executes the SCI operations outlined in Figure 1.3 within its designated CFP range, constructing indexes for previously saved and cross-client chunks. After duplicate detection, all these indexes are merged into a single index for use in container allocation.

Container allocation is performed next, and we increase its throughput through task parallelism. Multiple client uploads are processed simultaneously by assigning a client's recipe files to one of the n processes in a pool. The allotted cache size is divided equally across the processors. The cache holds the $CFP \rightarrow CID$ mapping pairs of newly allocated chunks. As the cache fills, it is sorted and then flushed to a client-specific overflow file. When all recipes for the client have been processed, a final sort and flush of the cache is performed.

After the container allocation has been completed for all client recipes, the overflow files from each client are loaded, merged and partitioned into one file per processor range. Using data parallelism again, a processor for each range updates the location of the newly allocated chunks into the SCI bins. If the new chunks would overflow the SCI bin, it is split.

4.4 Improved Caching

P-SCAIL improves SCAIL's caching technique described in Section 3.3 SCAIL Algorithm Stage 4. When the new-chunks cache fills, instead of making a full pass through the disk-based SCI index and writing the new chunks to the SCI bins, P-SCAIL sorts them and writes them to a single overflow file, as seen on lines 29-32 of **P-SCAIL Algorithm, Stage 4, Part 2** (see page 116). After processing all the "Missing" chunk recipes, the SCI process executes a final flush on lines 34-37.

In Part 2, as described above, the new chunk allocation locations are written to a single disk file, in sorted groups of $CFP \rightarrow CID$ mapping pairs each time the cache was flushed. To write these to the SCI, an additional algorithm is required. In **P-SCAIL Algorithm, Stage 4, Part 3** (see page 117), the sorted blocks of CFPs in the overflow file are sorted-merged, and P-SCAIL conducts a single update pass through the SCI bins.

This approach requires only a small number of CFP's to be held in memory (one for each sorted CFP block flushed during allocation) and ensures only a single write pass will be made through the SCI bins.

Since flushing the cache during container allocation is now a quick process, we can allow more cache flushes, which will allow us to reduce the cache size. This results in fewer overall disk I/Os and smaller memory requirements for P-SCAIL's SCI implementation compared to SCAIL.

P-SCAIL Algorithm, Stage 4, Part 2: Allocate to Containers in Recipe Order

/ Identical to SCAIL Algorithm, Stage 4, Part 2 (p. 82), except for highlighted areas. */*

Server input:
Duplicate Index DI, Cross-user Index CI from Stage 4, Part 1
On-disk sorted chunk index $SCI = ((fp, cid))$
Metachunk index MI, File Index FI
Memory cache persisting between calls $Cache = ((fp, cid))$
Metadata Containers, Data Containers
Receive from Stage 3:
Target file name, $MFP, EMK, UploadFP, UploadData$
Initialise empty $NewSCI$ for new sorted chunks locations
Initialise empty $NewMI$ for new metachunk locations
while fp in $UploadFP$ is a mfp **do**
    Retrieve tuple (mfp, emc, CFP_{mc}) from $UploadData$
    Initialise empty list of data container ids $CIDSForSegment = ()$
    while next fp in $UploadFP$ is a cfp **do**
        Retrieve pair $(cfp,\ ec)$ from $UploadData$
        if $cfp \in DI$ **then**
            Skip this chunk (duplicate)
        else if $cfp \in CI$ **then**
            Retrieve cid from $CI[cfp]$
            if $cid \neq nil$ **then**
                Add cid to $CIDSForSegment$
            else
                Allocate ec to data container
                Add (cfp, cid) to $Cache$
                Add cid to $CIDSForSegment$
            end if
        else
            Allocate ec to data container
            Add (cfp, cid) to $Cache$
            Add cid to $CIDSForSegment$
        end if
    end while
    Store $emc, CFP_{mc}, CIDSForSegment$ into current metadata container cid_{mc}
    Add (mfp, cid_{mc}) pair to $NewMI$
    if Cache is full **then**
        Append sorted $Cache$ as NEW_n to $NewSCI$
        Clear $Cache$
    end if
end while
if Cache is not empty **then**
    Append sorted $Cache$ as NEW_n to $NewSCI$
    Clear $Cache$
end if

P-SCAIL Algorithm, Stage 4, Part 3: Update Chunk and Metachunk Indexes

Server input:
On-disk sorted chunk index $SCI = ((cfp, cid))$
Metachunk index MI, file index FI
From **P-SCAIL Stage 4, Part 2**:
File with sorted groups of new chunk storage locations $NewSCI$
File with appended metadata index updates: $NewMI$
Receive from **P-SCAIL Stage 3**:
Target file name, MFP, EMK
Sort-merge all $NEW_1, NEW_2, \ldots, NEW_n$ from $NewSCI$ into a sorted stream NEW
Initialise iterators: $i \leftarrow 0$ for NEW, $j \leftarrow 0$ for SCI
while $i < |NEW|$ **and** $j < |SCI|$ **do**
    Find the next new chunk storage mapping: $(cfpNew, cidNew) \leftarrow NEW[i]$
    Find the next stored chunk storage mapping: $(cfpStored, cidStored) \leftarrow SCI[j]$
    if $cfpNew < cfpStored$ **then**
        Add $(cfpNew, cidNew)$ at $SCI[j]$
        if SCI bin full **then**
            Split bin
        end if
        Increment i, moving to the next new storage mapping
    else
        Increment j, moving to the next stored storage mapping
    end if
end while
for each $(mfpNew, cidNew)$ pair in $NewMI$ **do**
    Update MI: $MI[mfpNew] = cidNew$
end for
Save MFP, EMK under target file name in FI

4.5 Security Analysis

We adopt the same threat model for P-SCAIL as outlined for SCAIL (Section 3.9). Given that P-SCAIL is identical to SCAIL — aside from using multiple processors on a single server and an enhanced caching scheme — it inherits the security measures described in Section 3.10. Specifically, P-SCAIL offers the same protections against brute force attacks, as provided by MLE and DupLESS, as well as mitigations against side-channel attacks by avoiding cross-user client-side deduplication.

4.6 Limitations

P-SCAIL inherits the limitations of SCAIL, as detailed in Section 3.11. Additionally, it introduces further limitations resulting from its specific multiprocessor utilisation and improved caching approach, which are described below.

4.6.1 Processor Count Dependencies

In the P-SCAIL design, clients partition their sorted chunk fingerprint files based on the number of processors used for Stage 4. This approach complicates file management during upload and processing on the server, as it locks the client upload to a specific configuration of the server. If an adjustment to the processor configuration is needed, the clients must be informed beforehand to adjust their partitioning accordingly. This dependency creates rigidity and requires synchronisation between clients and servers. While the server may reprocess these files to match a changed processor configuration, we don't address this in our design.

4.6.2 Additional Storage For Cache-Backing Files

During Stage 4, rather than immediately flushing the cache of new $CFP \rightarrow CID$ (chunk fingerprint to chunk ID) mappings directly to the on-disk SCI bins, our design utilises

cache-backing files. This method avoids multiple processes simultaneously writing to the bins. It also enables all the mappings to be amalgamated and divided on a per-processor basis so that data parallelism can be used to update the SCI index.

However, this approach introduces the need for additional storage to accommodate these cache-backing files. Also, it slightly increases the window of vulnerability concerning data consistency. Specifically, there is a now a longer delay between when new mappings are created and when they are securely written to their final, on-disk chunk index location. On the other hand, since the cache is now file-backed, it may assist in the recoverability of the operation if it fails or is interrupted.

4.7 Evaluation

For multiprocessor experiments, we used the distributed framework Ray [60], limiting the number of CPUs available to the number required for each test. We also used an in-memory Redis database [16] to hold the 'previously stored' and 'cross-client' duplicate tables so multiple processes could access them. To update container allocations in the Redis cross-client table, we used an atomic update mechanism to avoid lost updates due to race conditions.

For the throughput experiments on the MS dataset, we created subsets of the volumes, starting from the first 16, and then repeatedly doubling the number of volumes up to the first 128, which comprised 43.2 TiB and 2.5 TiB after deduplication, also for a duplicate elimination ratio of 17.

By adjusting the number of processors and clients involved in backups, we explore SCAIL's metadata deduplication throughput under various scenarios.

4.7.1 Deduplication Throughput

We present measurements for both the client-side and server-side deduplication stages to assess the throughput performance of SCAIL. We define client-side and server-side

deduplication throughput as the total volume of data submitted by all clients for all backups, divided by the total wall clock time for all backups to complete Stage 2 and Stage 4, respectively.

We note that the reported throughput metrics are derived from trace datasets, which primarily capture the metadata manipulation aspect of deduplication. Server-side deduplication (Stage 4) throughput for complete (non-trace) datasets would be substantially lower due to the inherent overhead of transferring terabytes of data from client upload files to data containers. Nonetheless, as the client-side deduplication predominantly concerns metadata processing, we believe the throughput figures represent what one might expect in a real-data deployment scenario. While the server-side throughput metrics offer less direct applicability, they still provide insights into performance bounds for processing metadata in practical implementations.

We first discuss SCAIL (single processor) throughput, comparing it to Base, followed by evaluation results for P-SCAIL multiprocessor scenarios.

Single Processor Throughput SCAIL's client-side deduplication primarily involves loading recipes and performing queries into the memory-based MFP-only hash table. For the FSL dataset with eight clients and 115 backups, the total number of queries was 16.0 million on a hash table that grows to 350,000 elements, resulting in a throughput of 100.9 GiB/second. For the MS dataset with 128 clients over eight backups, which required 12.1 million queries on a hash table that grows to 2 million elements, the throughput was 77.9 GiB/second.

In contrast, Base and Metadedup conduct client-side deduplication through queries into memory-based CFP hash tables. These systems required a considerably higher number of lookups: 6.3 billion on a hash table, growing to over 50 million elements for the FSL dataset, and 5.8 billion lookups on a hash table, growing to over 350 million elements for the MS dataset. The single-processor client-side deduplication performance for Base against the FSL dataset was 1.9 GiB/second, and 1.3 GiB/second for the MS

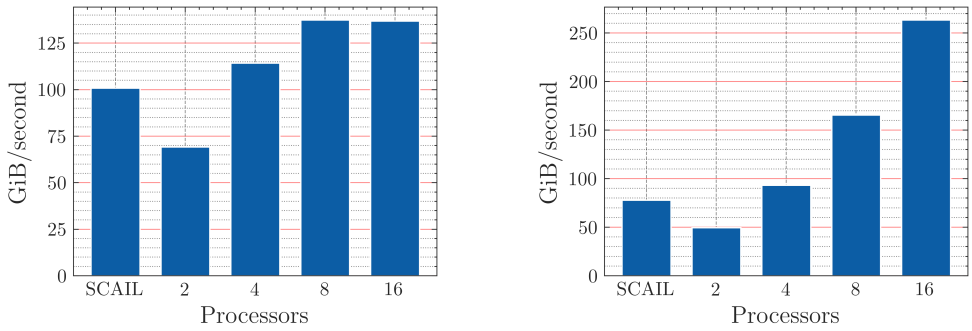
dataset. The throughput for Metadedup would be slower than Base, since it works with both MFPs and CFPs. Even with Base's memory-based hash table (which would be infeasible for large-scale workloads as outlined in Section 3.4), SCAIL's single processor client-side deduplication throughput is between 50 and 60 times faster.

P-SCAIL's server-side metadata deduplication performs the disk-based operations described in Section **3.3 SCAIL Stage 4**, resulting in 1.7 GiB/second throughput for the FSL dataset, 1.0 GiB/second for the MS dataset.

Base and Metadedup also assign new chunk data to containers in recipe order and update the memory-based chunk fingerprint index with CIDs of the new chunks. This memory-based operation enables server-side deduplication throughput of 56.1 GiB/second for the FSL dataset and 4.3 GiB/second for the MS dataset. The notably high throughput for the FSL dataset reflects that, on average, only 4 GiB of new data – specifically, data not filtered out with client-side deduplication – reaches the server for deduplication. In contrast, the MS dataset introduces an average of 350 GiB of fresh data with each backup. It's also worth noting that within the FSL dataset, there are numerous days, predominantly weekends, where no new data is generated. This absence of new data results in exceptionally high server-side deduplication throughput rates for Base on such days.

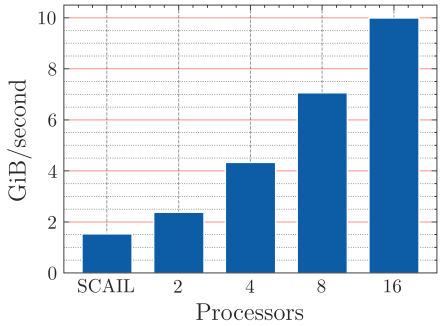
As previously noted, these server-side throughput figures are based on trace datasets. In real-world scenarios with non-trace dataset backups, all schemes' throughput would be substantially slower due to the extensive disk operations required for transferring client data.

P-SCAIL Multiprocessor Throughput Our multiprocessor evaluation comprises two separate experiments. In the first experiment, we repeatedly doubled the number of processors, holding the number of clients steady. In the second, we hold the processor count at 16 and repeatedly double the number of clients. Since the FSL dataset has only eight clients, we don't include it in the second experiment.

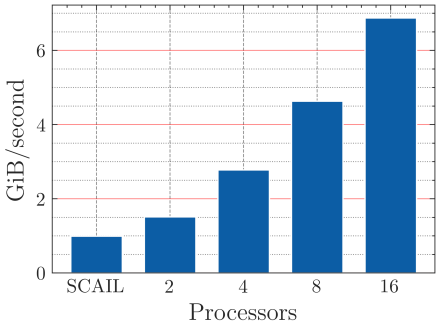


(a) FSL 8-clients, Client-side

(b) MS 128-clients, Client-side



(c) FSL 8-clients, Server-side



(d) MS 128-clients, Server-side

Figure 4.1: Client-side and server-side deduplication throughput for the FSL and MS datasets, comparing single processor SCAIL (first x-bar) and multiprocessor P-SCAIL (with 2, 4, 8 and 16 processors).

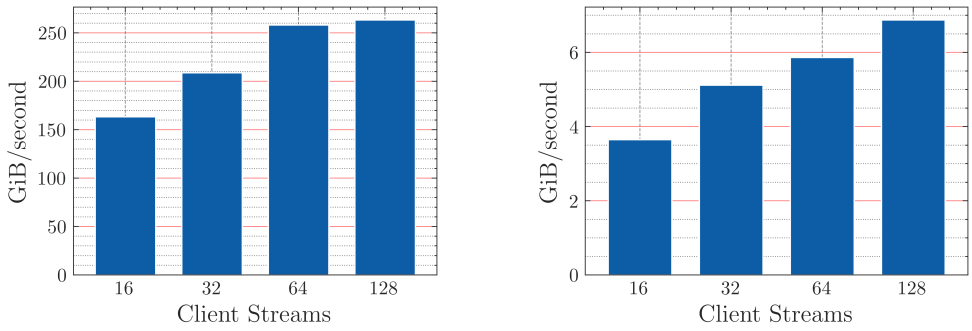
Experiment I. We assessed the effect of increasing the number of available processors using our two datasets: the long-term FSL dataset with eight clients over 115 backups, and the high-volume MS dataset with 128 clients across eight backups. For single processor throughput measurements, we employed SCAIL (after incorporating P-SCAIL's cache overflow method). Then we used P-SCAIL, progressively doubling the processors from 2 to 16.

The upper two bar charts of Figure 4.1 show client-side deduplication throughput for the FSL and the MS datasets. The first bar in each chart shows the throughput figures using the single processor SCAIL algorithm, enhanced with the cache management introduced in P-SCAIL. This approach avoids the need to start multiple processes for different phases of the backup process, eliminating the overhead of inter-process communication.

For the FSL dataset, the deduplication process takes approximately 5 seconds to deduplicate 500 GiB per backup. This short deduplication time means that overheads in P-SCAIL's multiprocessor scenarios, shown in the subsequent bars, reflect reduced per-processor throughput measurements compared to single-processor SCAIL. Moreover, in Figure 4.1a, it can be seen that there's no throughput increase beyond eight processors. This is because the FSL dataset only has eight clients, and client-side deduplication is task parallelised.

In contrast, for the MS dataset with 128 clients, throughput increases significantly as the number of processors increases. On average, throughput rises by 35%, with performance increasing from 77.9 GiB/second to 263.5 GiB/second as additional processors are added.

The lower two bar charts of Figure 4.1 illustrate the server-side deduplication throughput. The throughput of SCAIL once again benefits from the simplicity of its single-processor algorithm, achieving 1.5 GiB/second(FSL) and 1.0 GiB/second(MS). Nonetheless, with each doubling of processors, throughput improves by an average of 60%, culminating in 10 GiB/second (FSL) and 6.9 GiB/second (MS). When the number of



(a) MS 16-procs, Client-side

(b) MS 16-procs, Server-side

Figure 4.2: Client-side and Server-side Deduplication Throughput with 16 processes on the MS dataset. The number of client streams is repeatedly doubled from 16 to 128. As larger client counts are reached, processor load starts to become balanced, resulting in less idle time and greater throughput.

processors doubles from eight to 16 for the FSL dataset, it is interesting that throughput continues to rise even though the number of processors exceeds the eight clients. This demonstrates that SCI's data parallelism operations continue to increase throughput with increased processor availability.

Experiment II. For the second experiment, we fixed the processor count at 16 and repeatedly doubled the number of clients in the MS dataset from 16 to 128, with results shown in Figure 4.2. Even though we processed the largest streams first [17], throughput unexpectedly increased as client volumes doubled. We expected throughput to be reduced as more clients and backup volume were introduced. This increase was caused by a few clients with large volume streams that generated substantially more new data at each backup than the other backup streams. The extended client-side duplicate lookup and server-side container allocation processing times for these larger streams left up to 15 processors idle. However, as the client count approached 128, the combined processing duration of the smaller clients began to match or exceed that of the one or two larger client backups, leading to a more balanced workload and subsequent improvements in throughput.

Table 4.1: Segment Size Effect on Memory and Upload Size in P-SCAIL

	Segment Size	512KiB	1 MiB	2 MiB	4 MiB
FSL	**Index Memory (GiB)**				
	Base	1.385			
	P-SCAIL Client-side	0.057	0.018	0.010	0.006
	P-SCAIL Server-side	0.275	0.275	0.275	0.275
	Total P-SCAIL	**0.332**	**0.293**	**0.285**	**0.281**
	Memory Change	**-76.3%**	**-79.1%**	**-79.6%**	**-79.9%**
	Upload Volume (GiB)				
	Base	825			
	P-SCAIL	628	676	736	805
	Upload Change	**-23.9%**	**-18.1%**	**-10.8%**	**-2.5%**
MS	**Index Memory (GiB)**				
	Base	10.358			
	P-SCAIL Client-side	0.197	0.110	0.062	0.035
	P-SCAIL Server-side	0.549	0.549	0.549	0.549
	Total P-SCAIL	**0.746**	**0.659**	**0.610**	**0.584**
	Memory Change	**-92.8%**	**-93.6%**	**-94.1%**	**-94.4%**
	Upload Volume (GiB)				
	Base	4,992			
	P-SCAIL	5,019	5,177	5,397	5,679
	Upload Change	**0.5%**	**3.7%**	**8.1%**	**13.9%**

4.7.2 Memory Use and Upload Volume

To evaluate memory usage and upload volume, following Metadedup, we performed backup operations on the datasets with 512KiB, 1 MiB, 2 MiB, and 4 MiB segment sizes.

Base and Metadedup both used a single memory-based index on the server for Stage 2 (client-side deduplication) and Stage 4 (server-side deduplication). The Base index contained CFPs only, and Metadedup's index added MFPs, so it was 1% larger. P-SCAIL used a much smaller MFP-only memory-based index for Stage 2. For Stage 4, 128 MiB of memory was required for loading a single SCI bin. Previously, SCAIL used a 1.875 GiB write cache for the MS dataset to limit the number of full passes writing newly allocated chunk locations to the SCI bins. P-SCAIL's use of cache overflow files allowed us to reduce the cache to 128 MiB, a 93% reduction. The FSL cache remains

128 MiB, as used in SCAIL. In addition, the duplicate index table required 26 MiB for the FSL and 320 MiB for the MS datasets.

Table 4.1 presents the change in memory requirements and the difference in upload volume produced after storing all backups in each dataset.

The figures shown are the memory and disk-volume measurements taken after running all client backups for the FSL and MS Datasets.

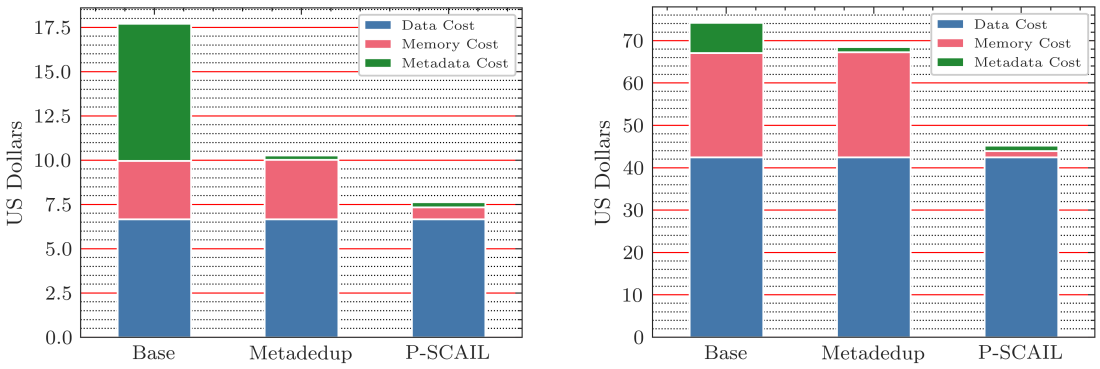
The first line in each group shows the volume of memory consumed by the CFP index used by the Base scheme. The "SCAIL Client-side" measurements reflect the volume of memory consumed by SCAIL's MFP-only index, and the "SCAIL Server-side" measurement reflects the volume SCAIL requires by using SCI to perform chunk-level deduplication on the server.

Upload volumes shown at the bottom of each group differ between Base and SCAIL because SCAIL deduplicates metadata, so it does not have to upload repeated File Recipes. These savings are offset required by RSD uploads in SCAIL, which increase as segment size increases.

As segments increase in size, there are fewer of them to store, decreasing the required memory size. P-SCAIL uses 76.3-79.9% less memory than Base for the FSL dataset and 92.9-94.4% less memory for the MS dataset.

4.7.3 Upload Overhead

P-SCAIL's use of coarse client-side deduplication caused it to upload redundant chunk data (see Section 3.8). For the FSL dataset, though, the 96-97% reduction (see Subsection 4.7.3) in metadata upload more than offset additional uploads of redundant chunk data. The outcome was a reduction in upload size compared to Base by 2.5-23.9%. For the MS dataset, which only had eight backups, there was an 86-88% decrease in metadata upload, resulting in an increased net upload size compared to Base of 0.5-13.8%.



(a) FSL dataset costs

(b) MS dataset costs

Figure 4.3: Deduplication processing and storage costs using the prices from Table 4.2 for the FSL and MS datasets. Base's metadata storage costs are reduced by metadata deduplication of File/Key Recipes, and memory requirements are further reduced by P-SCAIL.

Memory and Storage Costs

We evaluate the memory and storage costs of Base, Metadedup, and P-SCAIL, with results shown in Figure 4.3 using prices from Table 4.2. Both Metadedup and P-SCAIL utilise 2 MiB segments, as Metadedup [44] found this provided the best metadata savings for the FSL dataset.

Next, we detail the components making up the categories of Data, Memory, and Metadata.

- Data: All schemes employ exact deduplication, so long-term data storage costs on the HDD are identical.
- Memory: Base and Metadedup primarily incur costs from the chunk fingerprint lookup hash table. In contrast, P-SCAIL uses the much smaller metachunk fin-

Table 4.2: Memory and storage prices in US dollars from www.amazon.com gathered June 2023.

Type	Hardware	Price	Price per GiB
Memory	Crucial 16 GiB Single DDR - CT16G4SFD824A	$38	$2.3750
SSD	Western Digital 4TB Internal SSD - WDS400T2B0A	$230	$0.0617
HDD	Seagate 16TB HDD Exos - ST16000NM001G	$230	$0.0154

gerprint lookup hash, with added memory for SCI, including loading a single bin from disk, the cache for new chunk container allocation mappings and the duplicate chunk index.

- Metadata: SCAIL used an HDD for metadata, but with P-SCAIL, we've added an SSD. File and Key Recipes start on the SSD during the deduplication process, moving to the HDD for long-term storage. The SSD also provides storage for any memory-based indexes, cache overflow files and SCI bins. We sum the HDD and SSD metadata costs for the total metadata storage costs.

FSL Total Costs: For the FSL dataset, Base costs $17.70, Metadedup costs $10.26, resulting in savings of **42.2%** over Base. The P-SCAIL costs are $7.64, saving **56.9%** over Base.

MS Total Costs: For the MS dataset, Base has costs of $74.33, Metadedup is $68.51, a savings of **7.8%** over Base, and P-SCAIL is $45.22, a **39.2%** reduction compared to Base.

4.8 Summary

In this chapter, we presented P-SCAIL, which builds upon the metadata storage savings introduced by Metadedup [44], while dramatically reducing memory requirements, allowing us to increase scalability and significantly speed up client-side deduplication throughput. P-SCAIL also benefits from the low-memory, high concurrent client processing capacity of Sorted Deduplication[36], and we adapt that algorithm to support robust security provisions. This integrated design enables P-SCAIL to increase system capacity and throughput while significantly reducing required resources on the server. P-SCAIL also integrates data and task parallelism within its architecture, accelerating client lookup responses and amplifying the deduplication server's concurrent client processing capabilities.

Looking ahead, we are currently investigating reducing the re-upload of previously saved data chunks by incorporating resemblance techniques (see, e.g. [89, 104, 110, 87]).

Chapter 5

PR-SCAIL: Parallel Resemblance SCAIL

5.1 Introduction

In this chapter, we address the challenge of reducing Redundant Segment Data (RSD) upload volume, which is an inherited side effect of the SCAIL algorithm and retained in its parallel implementation, P-SCAIL. While P-SCAIL effectively harnesses parallel processing for performance gains, like SCAIL, its segment-based approach to client-side deduplication results in redundant uploads of some previously stored chunks.

To tackle this problem, we developed PR-SCAIL, which introduces a resemblance-based technique for memory-efficient chunk-level client-side deduplication. By leveraging the core strengths of SCAIL and P-SCAIL while addressing their limitations, PR-SCAIL minimises redundant uploads while retaining the performance benefits of parallelism.

We begin by reviewing the P-SCAIL algorithm to identify the origins and implications of RSD. Next, we detail the design and implementation of PR-SCAIL, outlining the enhancements made to reduce RSD. Finally, we compare PR-SCAIL with Resemblance Mergence Deduplication (RMD) [104], highlighting the advantages of our approach and its implications for encrypted deduplication backup servers.

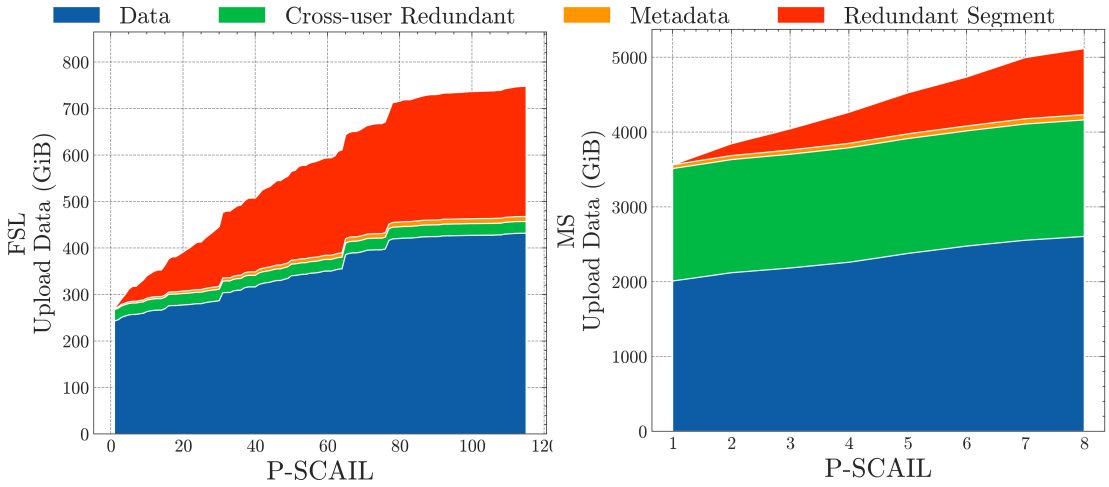


Figure 5.1: Breakdown of upload volume by component type for the FSL (left) and MS Dataset (right), using the P-SCAIL technique. P-SCAIL substantially reduces metadata upload volume, but introduces RSD upload volume.

5.2 P-SCAIL Overview

Recall that the P-SCAIL client encrypts chunks and generates the hash of the encrypted chunk in a CDC algorithm, which uses patterns in the hash values to determine segment boundaries. It then gathers the metadata of the chunks of the segment, including the chunk encryption keys, into a metachunk. The metachunk is then encrypted, and the hash of the encryption is taken. This hash value is a metachunk fingerprint that acts as a digest that uniquely identifies the segment. The MFP index holding metachunk fingerprints is used to perform client-side deduplication and identify segments that a client has previously stored so its constituent chunks do not have to be uploaded. The metachunks themselves are also deduplicated, reducing metadata storage on the server.

5.3 Redundant Segment Data Generation

Our previous designs, SCAIL and P-SCAIL, generate Redundant Segment Data (RSD) uploads, as shown in Figure 5.1. For a full discussion of these results, see Section 3.8.

These additional uploads are caused by P-SCAIL's segment-based client-side deduplication; that is, it sometimes uploads previously-stored chunks. Note that our definition of RSD excludes duplicate chunks from distinct clients in a given backup. Unlike RSD, this form of upload redundancy should *not* be eliminated, because doing so would expose the system to side-channel attacks, where clients might be able to identify chunks that other clients have uploaded.

To understand where RSD may occur, consider the situation of a modification of a small number of chunks within a segment on a client's machine, from one backup to the next. Since CDC is used to create the segment boundaries, if the modifications are minor, the segment boundary will remain the same. Another scenario is where the final chunk of the segment is modified, which will likely extend the segment and shorten the following segment unless the following segment is also extensively modified. But in P-SCAIL, *any* change of a segment chunk will create a distinct metachunk fingerprint, triggering P-SCAIL to upload all chunks of the segment. In this scenario, all chunks except the modified chunks will be Redundant Segment Data (RSD), as described in Section 3.8.

We therefore categorise some of the causes of RSD between backup generations as follows:

1. Segments that have had a small number of chunks modified but retain their segment boundaries.

2. Segments that have been sufficiently modified that they generate new segment boundaries.

3. Wholly new segments in the system that coincidentally contain one or more chunks uploaded previously. Systems with specific commonly occurring chunks (e.g., chunks containing all zeroes or all spaces) will more likely generate new segments with these common, previously uploaded chunks.

Having identified possible sources of RSD, we discuss our attempts to reduce it.

5.4 Reducing Redundant Segment Data

Using a chunk-based index for client-side deduplication would eliminate RSD, but P-SCAIL explicitly avoids this because it does not scale to large datasets as shown in Subsection 2.3.1. To retain P-SCAIL's scaling capabilities, we need a method of efficiently identifying previously saved chunks that requires a low amount of memory.

Suppose we can identify the segment containing the previously saved chunks that produced the RSD? In that case, we should be able to eliminate many of the previously saved chunks, giving high, although not always exact, deduplication efficiency. As shown by Resemblance Mergance Deduplication (RMD) [104], segments that resemble each other can be identified by adapting Broder's *document* resemblance to *segment* resemblance. Citing from the RMD abstract (the following quotation is italicised):

> *At data ingesting time, RMD uses a resemblance algorithm to detect resemble data segments and put resemblance segments in the same bin. As a result, at querying time, it only needs to search in the corresponding bin to detect duplicate content, which significantly speeds up the query process.*

RMD and many other approaches [9, 92, 89, 106] use the data resemblance theory advanced by Broder [14, 13] that the probability of two documents sharing the same minimum hash value is highly dependent upon their similarity degree. From Section 2.2 in [89] quoting from Broder (the following quotation is italicised):

Consider two sets S_1 and S_2 with $H(S_1)$ and $H(S_2)$ being the corresponding sets of the hashes of the elements of S_1 and S_2 respectively, where H is chosen uniformly and randomly from a min-wise independent family of permutations. Let $min(S)$ denote the smallest element of the set of integers S. Then

$$Pr\left[min(H(S_1)) = min(H(S_2))\right] = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}.$$

5.5 Design of PR-SCAIL

PR-SCAIL makes use of resemblance to reduce RSD. This involves modifications to each of the four stages in P-SCAIL. In Stage 1, in addition to building recipes of MFPs, the list of CFPs making up the MFP will be included, and one CFP (the minimum valued CFP) is selected as the Representative Fingerprint (RFP) for each segment. During Stage 2 (the lookup stage), the server adds and must maintain an RFP index, which is used to find any previously saved similar segments used for chunk deduplication. Stage 3 under P-SCAIL receives only a list of "Missing" MFPs and must upload all chunks making up the MFP. In PR-SCAIL, only the CFPs for chunks not deduplicated are specified, so only these chunks must be uploaded. In Stage 4, there are no deduplication changes for PR-SCAIL; the usual cross-user, chunk-level deduplication is performed, resulting in exact deduplication. As the last step in Stage 4, the MFP and new RFP index are updated for all new segments.

5.6 Modifying P-SCAIL for PR-SCAIL

This section outlines the key modifications necessary to transition from the P-SCAIL system to PR-SCAIL. We detail how this enhancement improves chunk-level resemblance deduplication while maintaining the core functionality of the Metachunk Fingerprint (MFP) Index. The following subsections explore these changes, highlighting their impact on the system's deduplication process and memory requirements.

5.6.1 Stage 1: Building the Lookup Query

Clients must upload not only the MFP fingerprints but also the segment recipes, which is a list of CFPs, one of which is designated as the RFP. Using CFPs and RFPs enables the server to perform chunk-level resemblance deduplication.

The client selects the RFP for each segment as the minimum CFP within the seg-

PR-SCAIL Algorithm, Stage 1: Build Deduplication and Resemblance Query

1: **Client input:** Target file name, client's key key, segment size S_{size}, chunk size C_{size}
2: Generate and store $MC, MK, PEMC, MFP$ as in **P-SCAIL Stage 1 (p. 76)**
3: Calculate resemblance recipes $RR = (rr_1, rr_2, \ldots, rr_m)$, where, for j in $[1, m]$:
$rr_j = (mfp_j, pr_j, rfp_j)$,
$mfp_j = MFP[j]$,
$pemc = PEMC[j]$
Extract (pr_j, emc_j) from $pemc$
$rfp_j = \min(pr_j)$
4: Send resemblance recipes RR to server

ment. That is, it selects the SHA-1 hash value with the lowest value when interpreted as a 20-byte number.

5.6.2 Stage 2: Lookup for Deduplication

Recall that a list of the constituent CFPs of segments are stored in the manifests of metadata containers, see Subsection 3.10.1. For PR-SCAIL, we reduce the expected metadata container size from 128 MiB to 4 MiB. This gives us much more discrimination in loading pertinent metadata manifests. Given an RFP, for resemblance deduplication, we must find the CID of the similar segment's metadata container and retrieve its CFPs from its manifest. The RFP index will hold this association.

Each segment will have an RFP, although by design, other segments may have this same RFP. The number of RFP entries in the index will therefore be at most the number of MFPs. For a PB of unique data using 2 MB segments, there would be at most 500 million RFPs. However, multiple segments will often be saved over time with the same RFP. To preserve memory, the system could hold only the segment's first or most recent version. PR-SCAIL aims for a high level of deduplication and so maintains a set with all of the CIDs of segments that have been backed up with the same RFP.

To minimise the memory and processing overhead of storing and comparing RFPs, we take a non-cryptographic hash of the RFP using the xxHash-64 algorithm referenced in [18], producing a 64-bit (8-byte) hash. This reduces the memory required to hold the RFPs but will increase the chance of a hash collision. Since PR-SCAIL maintains a list of all CIDs that have ever been seen for each RFP, a collision will still result in deduplication between all previously stored similar segments, but may also include some dissimilar segments. The net effect of a collision would be the same deduplication efficiency but at the cost of loading chunk fingerprints from dissimilar segments during the deduplication operation.

The lookup stage in the PR-SCAIL deduplication process involves a more complex lookup operation than P-SCAIL. First, identical segments are eliminated using the MFP index. This step is pivotal to PR-SCAIL's performance, as it effectively eliminates almost all segments in a backup workload, thereby isolating only the new or modified

PR-SCAIL Algorithm, Stage 2: Deduplication Lookup with Resemblance

Server input:
Metachunk index MI
Resemblance index $RI = ((rfp, CIDList))$
Receive: resemblance recipe RR from **PR-SCAIL Stage 1**
Initialise:
MS, a list of (mfp, rfp) for missing segments
$CFP_{client} = \{\}$, a set to hold CFPs from client for deduplication
MR, a list of missing metachunks and chunks after resemblance deduplication
for each resemblance recipe (mfp, pr, rfp) in RR **do**
    if $mfp \in MI$ **then**
        Remove (mfp, pr, rfp) from RR
    else if then
        Add CFPs of pr to set CFP_{client}
    end if
end for
for each remaining resemblance recipe (mfp, pr, rfp) in RR **do**
    if $rfp \in RI$ **then**
        Retrieve list of container IDs: $CIDList = RI[rfp]$
        for each container ID cid in $CIDList$ **do**
            Load chunk fingerprints CFP_{cid} from metadata container cid
            Remove all CFPs in CFP_{cid} from CFP_{client}
        end for
    end if
end for
for each remaining resemblance recipe (mfp, pr, rfp) in RR **do**
    Initialise list $pr_{remaining} = ()$
    for each cfp_i in pr **do**
        if $cfp_i \in CFP_{client}$ **then**
            Add cfp to $pr_{remaining}$
        end if
    end for
    if $CFP_{remaining}$ is not empty **then**
        Add $(mfp, pr_{remaining}, rfp)$ to MR
    end if
end for
Send MR to the client

segments for further processing. Next, for these remaining segments, we:

1. Build a set of all CFPs in the lookup recipes from the client.
2. Using the RFP index, build a set containing the CID recipe storage location for any segment this client has previously saved that is similar to the segments of this upload.
3. Search the CIDs with the most references first. Create a sorted list of the CIDs in ascending order on the number of segments that reference the CID. Performing the search in this sorted order increases the chance that all CFPs have been deduplicated without loading all CIDs, so the search can be terminated early, as outlined in Step 6.
4. Process each CID in the sorted list, loading the set of CFPs from the manifest of the metadata container.
5. Deduplicate using set difference: remove the set of loaded recipe CFPs from the set of CFPs built in step 1.
6. If all CFPs from the recipes have been eliminated, the rest of the CIDs do not have to be examined.

It should be noted that deduplicating a large quantity of CFPs is accomplished through a single access to the manifest section of the metadata container. This process facilitates the deduplication of hundreds of segments and tens of thousands of CFPs in a single read and set difference operation. Also, the total number of retrievals is limited to the number of metadata containers that the client has stored.

Next, the segment recipes uploaded from the client are processed to produce the "Missing" recipes for Stage 3. Each CFP in the segment recipes is looked up in the deduplicated CFPs set. If any CFPs are found in a segment, they are output associated with the MFP of the segment. Note that for efficiency, instead of returning a list of the

"Missing" CFPs, we output just the list of *offsets* for CFPs in the recipe list instead of a list of CFPs. For instance, if the only remaining chunks in a segment are the 2nd and 3rd chunks in the segment recipe, we'll return the zero-based offsets into the recipe, for example "[1, 2]". Since the client builds the recipe, it knows which CFPs are being referred to.

Stage 2 directly benefits from the established task-based parallelism framework of P-SCAIL. Specifically, lookup for the deduplication process, now enhanced by chunk-level analysis, is processed independently in a separate task, allowing PR-SCAIL to maintain high efficiency and throughput. Still, with near-exact client-side deduplication, there will be some duplicate chunks that will not be identified, and so will be uploaded to the server. PR-SCAIL identifies and eliminates any of these remaining duplicate chunks during cross-user server-side deduplication and produces exact deduplication of chunks stored on the server.

5.6.3 Stage 3: Assemble Missing Metachunks and Chunks

PR-SCAIL Algorithm, Stage 3: Assemble Missing Metachunks and Chunks

/ Identical to SCAIL Stage 3 (p. 79), except for highlighted areas. */*

Client input:
- Target file name, client key *key*
- $MFP, MC, PEMC, MK, EC$ from **SCAIL Stage 1** (p. 76)

Receive: Missing resemblance list MR from **PR-SCAIL Stage 2** (p. 137)

Initialise:
- Upload data list: $UploadData = ()$
- List of unique MFPs and their CFPs in recipe order: $UploadFP = ()$

for (mfp, pr, rfp) in MR **do**
  if $mfp \notin UploadFP$ **then**
    Find index j such that $MFP[j] = mfp$
    Retrieve partially encrypted metachunk $pemc = PEMC[j]$
    Append $(mfp, pemc)$ to $UploadData$
    Add pair (mfp, rfp) to $UploadFP$
    for each *cfp* in $CFP_{remaining}$ **do**
      if $cfp \notin UploadFP$ **then**
        Find index i such that $CFP[i] = cfp$
        Retrieve encrypted chunk $ec = EC[i]$
        Append pair $(cfp,\ ec)$ to $UploadData$
        Add *cfp* to $UploadFP$
      end if
    end for
  end if
end for

Extract and sort CFPs from $UploadFP$ into $SortedCFP$

Encrypt metachunk key recipe: $EMK = \text{Encrypt}(MK,\ key)$

Send target file name, $MFP, EMK, UploadFP, UploadData, SortedCFP$ to server

For all MFPs returned in the "Missing" recipes from Stage 2, P-SCAIL would include all chunks in the segments in the assembled upload. Since the PR-SCAIL "Missing" recipe includes specific CFPs, only these chunks will be assembled for upload, achieving near-exact, chunk-level client-side deduplication.

5.6.4 Stage 4: Cross-User Deduplication

To mitigate side-channel attacks, all SCAIL family algorithms *do not* perform cross-client deduplication on the client. Any cross-client duplicate data or previously uploaded chunks missed using the resemblance technique will be eliminated during chunk-level deduplication on the server. This novel approach results in exact deduplication storage for PR-SCAIL, even though client-side deduplication is near-exact since it is resemblance-based.

Finally, note that the segment resemblance is not performed during the system's first backup. This is because carrying out segment resemblance at this stage would have no effect, given that no resemblance data has been generated yet. Consequently, it is only implemented in the subsequent backups, where RFPs are available for similarity detection.

PR-SCAIL Algorithm, Stage 4, Part 2: Allocate to Containers in Recipe Order

/ Identical to P-SCAIL Stage 4 Part 2 (p. 116), except for highlighted areas. */*

Server input:
Duplicate Index DI, Cross-user Index CI from Stage 4, Part 1
On-disk sorted chunk index $SCI = ((cfp_1, cid_1), (cfp_2, cid_2), \ldots, (cfp_n, cid_n))$
Metachunk index MI, File Index FI
Persisting cache $Cache = ((cfp_1, cid_1), (cfp_2, cid_2), \ldots, (cfp_n, cid_n))$
Metadata Containers, Data Containers
Receive: target file name, $MFP, EMK, UploadFP, UploadData, SortedCFPs$
Initialise empty $NewSCI, NewMI,$ NewRI for new container mappings
while fp in $UploadFP$ is a pair(mfp, rfp) **do**
    Retrieve (mfp, emc) from $UploadData$
    Initialise empty list of data container ids $CIDSForSegment = ()$
    while next fp in $UploadFP$ is a cfp **do**
        Retrieve pair (cfp, ec) from $UploadData$
        if $cfp \in DI$ **then**
            Skip this chunk (duplicate)
        else if $cfp \in CI$ **then**
            Retrieve cid from $CI[cfp]$
            if $cid \neq nil$ **then**
                Add cid to $CIDSForSegment$
            else
                Allocate ec to data container
                Add (cfp, cid) to $Cache$
                Add cid to $CIDSForSegment$
            end if
        else
            Allocate ec to data container
            Add (cfp, cid) to $Cache$
            Add cid to $CIDSForSegment$
        end if
    end while
    Store $emc, CFP_{mc}, CIDSForSegment$ into current metadata container cid_{mc}
    Add (mfp, cid_{mc}) pair to $NewMI$
    Add (rfp, cid_{mc}) pair to $NewRI$
    if Cache is full **then**
        Append sorted $Cache$ as NEW_n to $NewSCI$
        Clear $Cache$
    end if
end while
if Cache is not empty **then**
    Append sorted $Cache$ as NEW_n to $NewSCI$
    Clear $Cache$
end if

PR-SCAIL Algorithm, Stage 4, Part 3: Update Chunk and Metachunk Indexes

/ Identical to P-SCAIL Stage 4 Part 3 (p. 117), except for highlighted areas. */*

Server input:
On-disk sorted chunk index $SCI = ((cfp_1, cid_1), (cfp_2, cid_2), \ldots, (cfp_n, cid_n))$
Metachunk index MI, file index FI
$NewSCI, NewMI,$ NewRI from **PR-SCAIL Stage 4, Part 2**
Receive target file name, MFP, EMK from **PR-SCAIL Stage 3**
Sort-merge all $NEW_1, NEW_2, \ldots, NEW_n$ from $NewSCI$ into a sorted stream NEW
Initialise iterators: $i \leftarrow 0$ for NEW, $j \leftarrow 0$ for SCI
while $i < |NEW|$ **and** $j < |SCI|$ **do**
    Find the next new chunk storage mapping: $(cfpNew, cidNew) \leftarrow NEW[i]$
    Find the next stored chunk storage mapping: $(cfpStored, cidStored) \leftarrow SCI[j]$
    if $cfpNew < cfpStored$ **then**
        Add $(cfpNew, cidNew)$ at $SCI[j]$
        if SCI bin full **then**
            Split bin
        end if
        Increment i, moving to the next new storage mapping
    else
        Increment j, moving to the next stored storage mapping
    end if
end while
for each $(mfpNew, cidNew)$ pair in $NewMI$ **do**
    Update MI: $MI[mfpNew] = cidNew$
end for
for $(rfpNew, cidNew)$ pair in $NewRI$ **do**
    Update RI: Add $cidNew$ to $CIDList$ at $RI[rfpNew]$
end for
Save MFP, EMK under target file name in FI

5.7 Comparison of PR-SCAIL with the RMD design

"RMD: A Resemblance and Mergence Based Approach for High Performance Deduplication" [104], described in the Related Works Subsection 2.7, differs from PR-SCAIL in the following ways:

1. RMD uses fixed-size segments, PR-SCAIL uses variable-sized segments, which increases the likelihood of producing the same segments even when modifications are made upstream of the segment.

2. RMD deduplicates the CFPs of recipes serially, in recipe order, loading all CFPs in the bin file of similar segments into an LRU cache. If the backup workload is large, the CFP cache will become ineffective, causing many cache misses. With each cache miss, a disk I/O to load the segment bin will be required, often triggering an additional cache eviction write.

3. RMD has to perform a disk write when segment bins are evicted from the LRU cache, updating their reference statistics and, if necessary, pruning the chunk lists stored in the bin. PR-SCAIL does not keep or update statistics for similar segment usage.

4. RMD has no mechanism to skip chunk-level deduplication on identical, previously saved segments. Even if there are no changes to the segment, if its chunks are not in the cache, it must load them from the disk and perform duplicate checks on all chunks in the segment. The SCAIL family of algorithms uses the MFP index to deduplicate an entire segment with a single memory-based lookup.

5. RMD maintains CFP reference counts stored in the segment bin files. It uses these statistics to potentially limit the number of CFPs loaded to deduplicate a segment. PR-SCAIL always loads all CFPs in a given CID record in one operation.

6. RMD gathers chunks from similar segments to a single disk file, which requires a disk read for each segment unless it is already in its cache. PR-SCAIL stores the recipes for hundreds of segments in the manifest of each metadata container record and so can load hundreds of thousands of chunks, describing more than 1 GiB of data, with a single manifest read operation.

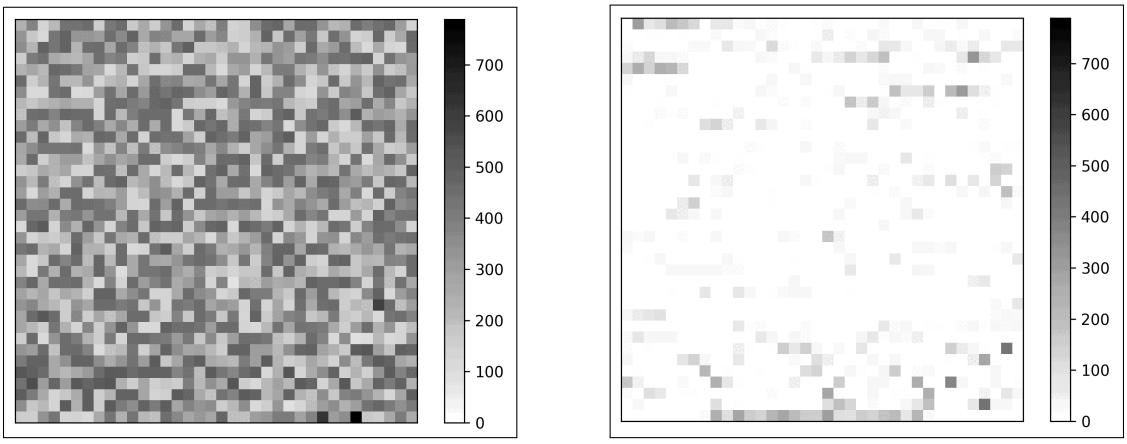
5.8 Security Analysis

We assume the same threat model for PR-SCAIL as SCAIL and P-SCAIL (see Section 3.9). Additionally, PR-SCAIL inherits the data privacy guarantees of SCAIL (see Section 3.10). However, since the client-side deduplication queries and replies from the server are performed at the chunk-level rather than the metachunk level, PR-SCAIL does not have the extra level of side-channel attack resistance that SCAIL provides. It aligns with those protections outlined in Metadedup (see Subsection 2.5.7).

5.9 Limitations

PR-SCAIL inherits the limitations of both P-SCAIL (Section 4.6) and SCAIL (Section 3.11), except for reducing the write amplification limitation. This is because PR-SCAIL performs similarity-based chunk-level client-side deduplication, that eliminates between 80% and nearly 97% of excess uploads on our evaluation datasets (Subsection 5.10.3), substantially mitigating the write amplification limitations present in SCAIL and P-SCAIL.

Since client-side deduplication in PR-SCAIL requires disk I/O operations to support chunk-level deduplication, its throughput is significantly slower than P-SCAIL's throughput (see Subsection 5.10.5). This may limit PR-SCAIL's ability to keep up with high volumes of client data capacity that the SCAIL family of algorithms enables. See Subsection 6.9.2 for a comparison of the trade-off between throughput and upload volume in P-SCAIL and PR-SCAIL.



(a) Before deduplication (b) After deduplication

Figure 5.2: Heatmaps reflecting the count of previously stored chunks in uploads before and after segment-based resemblance deduplication.

5.10 Evaluation

We evaluate PR-SCAIL using the same methodology and experimental setup as P-SCAIL (as detailed in Section 4.7). We begin by examining the efficacy of the resemblance technique in diminishing Redundant Segment Data (RSD) upload volume in an example client backup. This analysis is extended to full datasets using the 2MiB segment size, and subsequently, we explore the technique's behaviour across a range of segment sizes from 512 KiB to 4 MiB. Following this, we delve into how varying segment sizes influence memory usage and the volume of data uploaded. Next, we examine the throughput measurements for various numbers of processors and clients. Finally, we will conclude this chapter by discussing the broader implications of our results.

5.10.1 Illustrating The Efficiency of Segment-based Resemblance Deduplication

In this subsection, we explore the effectiveness of the segment resemblance technique on an individual backup for a single client. Figure 5.2, shows two heatmaps illustrating the outcome of segment resemblance deduplication. The values in each heatmap ele-

ment denote the number of chunks within "modified" segments—defined as segments where this specific sequence of chunks (as identified by the segment's metachunk) has not been saved before and contains less than 100% new chunks. The heatmap on the left illustrates chunk counts per segment prior to deduplication, whereas the right heatmap depicts the counts after deduplication.

To represent the data visually, we calculated the dimensions of each square-shaped heatmap based on the total number of elements. By taking the square root of the total number of modified segments, we determined the size of each side of the square. If the number of segments was not a perfect square, we padded the final row of the heatmap with zero counts.

We then transformed the list of chunk counts into a two-dimensional array, suitable for heatmap visualisation. This involved reshaping the data into rows and columns, correlating with the square root value calculated for the heatmap dimensions.

For the colour mapping, we employed a greyscale where different shades represent varying chunk counts. A light shade indicates a low number of chunks, aiding in the visual differentiation of segment densities before and after deduplication. This shade scale aids in interpreting the changes in chunk distribution due to the deduplication process.

These segments are prime candidates for containing previously uploaded chunks, which our technique aims to minimise. Initially, the left heatmap reveals a wide range of chunk counts in modified segments, averaging 316 chunks per segment. Post-deduplication, as shown in the right heatmap, this average is significantly reduced to 26 chunks per segment.

This specific analysis involves data from the 5th backup of Client 7 in the FSL dataset, dated 2013-01-27, following a backup on 2013-01-26. The client's backup query consists of 12,313 MFPs describing 32.6 GiB of data across 3,582,414 CFPs. Upon examining the MFP index, we find that 11,020 segments, encompassing 3,172,260 CFPs, had been previously saved and are therefore excluded from further analysis. The re-

Table 5.1: Segment Size Effect on Redundant Segment Data Volume

	Segment Size	512KiB	1 MiB	2 MiB	4 MiB
FSL	P-SCAIL Redundant Segment (GiB)	170.1	219.4	279.9	348.4
	PR-SCAIL Redundant Segment (GiB)	8.4	8.4	8.4	8.4
MS	P-SCAIL Redundant Segment (GiB)	599.1	754.3	964.0	1,248.0
	PR-SCAIL Redundant Segment (GiB)	183.6	183.6	183.6	183.6

maining data includes 1,303 segments with 410,154 CFPs, equating to 3.5 GiB. Seven segments are identified with 100% new chunks, reducing the count to 1,296 "modified" segments. Through the process of segment resemblance deduplication, 379,229 chunks (92.5%) are eliminated from these segments. Consequently, the average number of chunks per segment in the heatmaps is markedly lowered from 316 to 26.

5.10.2 Reducing RSD Volume for various segment sizes

In Table 5.1, we show the Redundant Segment Data (RSD) upload volumes in GiB that accumulate after performing all backups in the FSL (top figures) and MS (bottom figures). We start our analysis at segment size 512KiB, and double the segment size until reaching 4 MiB segment size.

For the P-SCAIL method, as the segment size increases, RSD significantly increases. This can be attributed to the fact that modifying even a single chunk in a larger segment results in a greater number of chunks being uploaded that were saved during a previous backup. For instance, with the MS dataset, the volume of RSD for P-SCAIL more than doubles as the segment size goes from 512 KiB to 4 MiB.

With PR-SCAIL, around 183 GiB of RSD still remains after segment resemblance deduplication, but importantly, it does not grow with the increase in segment size. This reflects the stable behaviour of PR-SCAIL deduplication since it is performed at the chunk and segment levels.

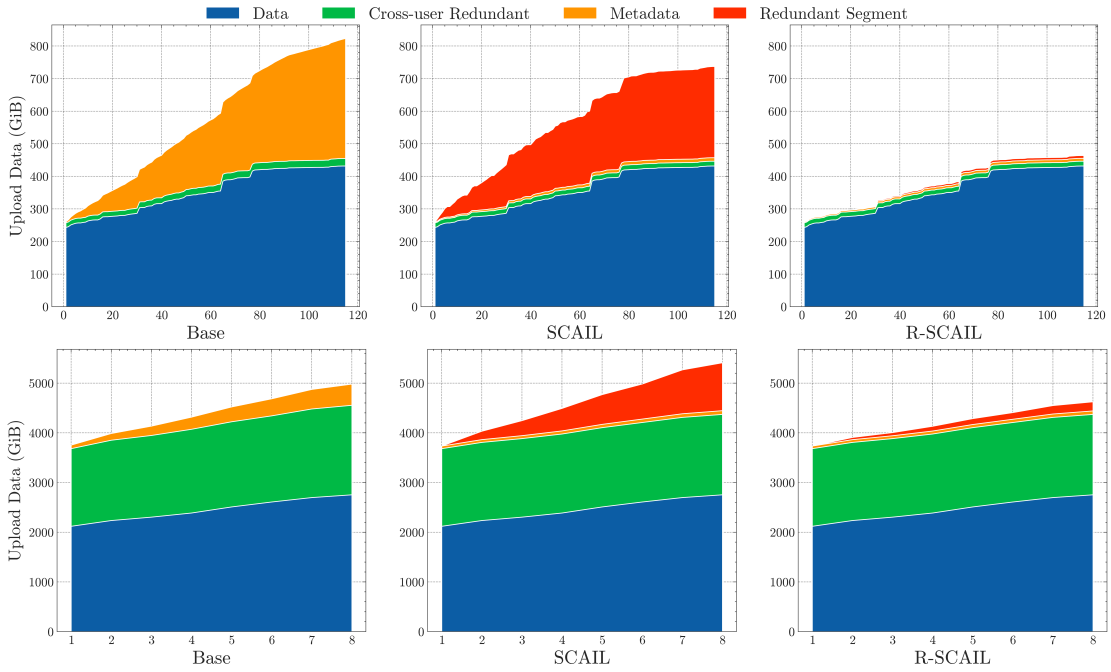


Figure 5.3: Breakdown of upload cumulative volume by component type for the FSL (top) and MS (bottom) datasets for the Base, P-SCAIL and PR-SCAIL techniques.

5.10.3 Total Upload Volume

The cumulative upload volume incurred running all backups for our two real-world datasets, FSL and MS, is illustrated in Figure 5.3. We evaluate three approaches: Base, P-SCAIL and the technique introduced in this chapter, PR-SCAIL. In this upload overhead evaluation, all approaches use 8KiB expected chunk size, and P-SCAIL and PR-SCAIL use 2MiB expected segment size.

In Stage 3 of the deduplication process, clients are responsible for uploading all encrypted chunk data that has not been filtered out during the client-side deduplication of Stage 2. The uploaded data volumes collected for each backup are categorised as follows:

1. **Data**: This category encompasses all uploaded unique encrypted chunks in all client uploads which have not previously been stored. All schemes, including Base, P-SCAIL and PR-SCAIL upload identical volumes of this type of data.

2. **Cross-user Redundant Data**: This includes data in the current upload batch that was either uploaded by another client in the same backup cycle or in a previous backup, but is not associated with the current client's prior uploads. All three schemes upload this type of data to protect against side-channel attacks.

3. **Metadata**: This pertains to file and key recipes that clients generate and save for data restore operations. Base's recipes are references to CFPs, while P-SCAIL and PR-SCAIL recipes are MFPs.

4. **Redundant Segment Data** (*P-SCAIL and PR-SCAIL only*): This category is assigned to any chunks in the current upload that have not been classified in the above categories and are identified as duplicates of chunks that are already stored within the system. These chunks tend to be the result of modification of segments between backups.

Base Algorithm: The Base algorithm employs a chunk index that maintains ownership information, enabling client-side deduplication within individual client scopes. It implements a strict privacy protocol to protect against side-channel attacks by acknowledging only those chunks previously uploaded by the active client. This ensures data uploaded by other clients remains undisclosed.

On the server side, it performs cross-user chunk-level deduplication. This achieves exact duplication across users without compromising the confidentiality of upload status. Within the FSL dataset — home directories from a small research lab — cross-user data is minimal. Over the backup timeline, the metadata volume increases steadily, eventually approaching the total size of unique data stored, even without significant new data additions, as seen in the top left chart in Figure 5.3.

By contrast, the MS dataset, shown in the bottom left chart, encompasses complete hard drive backups from a corporate setting. It exhibits substantial cross-user data in the initial backup. The Base system establishes ownership for this cross-user data upon

Table 5.2: Total upload volume and the difference in upload volume between Base and PR-SCAIL

	Segment Size	512KiB	1 MiB	2 MiB	4 MiB
FSL	Base (GiB)	823.5			
	PR-SCAIL (GiB)	465.5	464.3	464.0	464.2
	Difference	**-43.6%**	**-44.8%**	**-43.8%**	**-43.7%**
MS	Base (GiB)	4,981.8			
	PR-SCAIL (GiB)	4,620.6	4,622.5	4,625.2	4,628.9
	Difference	**-7.4%**	**-7.4%**	**-7.4%**	**-7.3%**

its first upload, effectively excluding such chunks from future uploads. Consequently, in later backups, there is only a marginal increase in cross-user data in the MS dataset.

Although the volume of metadata increases with each backup, there are only eight backups in total. As a result, the overall metadata remains small compared to the total upload size.

P-SCAIL Metadata Deduplication: P-SCAIL significantly lowers the server's metadata storage requirements by storing and deduplicating metachunks. However, its use of segment-level deduplication (for example, using 2MiB segments) leads to the upload of redundant segment data (RSD), as seen in the top middle chart in Figure 5.3. Fortunately, server-side deduplication efficiently eliminates all RSD before incorporating it into the server's unique data store.

Within the FSL dataset, we observe a marked reduction in metadata volume because of metadata deduplication. However, the volume of RSD upload gradually begins to rival that of unique data upload volume.

In the MS dataset in the bottom middle chart, P-SCAIL reduces the metadata uploads. Still, the accumulation of uploaded RSD surpasses the volume of saved metadata uploads, resulting in a net increase in upload volume compared to Base.

PR-SCAIL Resemblance Deduplication: PR-SCAIL retains P-SCAIL's metadata deduplication, and adds resemblance-based client-side deduplication. It identifies segments similar to the segments uploaded by clients, loads the recipes for those similar seg-

ments and performs chunk-level deduplication. In the FSL dataset, this proves very effective and eliminates 96.99% of RSD, as evidenced in the top right chart in Figure 5.3.

A summary of the upload volumes for all types of upload, including data, metadata, and excess uploads (see Subsection 3.8) for the Base and PR-SCAIL schemes are shown in Table 5.2. Segment sizes in PR-SCAIL are sampled, ranging from 512 KiB to 4 MiB. The difference in upload volume between the two schemes is also shown.

This results in a 43.8% reduction in upload volume for PR-SCAIL vs. Base. In the MS dataset, 80% of the RSD is eliminated, resulting in 7% reduction in upload volume vs Base, as seen in the bottom right chart in Figure 5.3.

Table 5.3: Effect of Segment Size on Memory Usage for Metachunk Fingerprint (MFP) and Representative Fingerprint (RFP) Indexes. Index sizes in MiB of RAM, with the number of index elements in parenthesis.

Segment Size		512KiB	1 MiB	2 MiB	4 MiB
FSL	MFP Index	31 MiB (1,093,664)	17 MiB (596,661)	9 MiB (330,308)	5 MiB (184,731)
	RFP Index	12 MiB (738,216)	6 MiB (373,267)	3 MiB (189,981)	2 MiB (97,213)
MS	MFP Index	198 MiB (6,927,263)	110 MiB (3,831,850)	61 MiB (2,135,874)	34 MiB (1,199,526)
	RFP Index	101 MiB (4,838,330)	52 MiB (2,445,329)	27 MiB (1,244,440)	14 MiB (636,707)

5.10.4 Memory Requirements

We evaluate how much additional memory is required by PR-SCAIL with the introduction of the RFP Index, in comparison to its predecessor, P-SCAIL. While both systems utilise the same MFP Index, PR-SCAIL's introduction of the RFP Index slightly elevates the overall memory footprint. Table 5.3 which details the memory size in MiB and the count of elements in "()" for both the MFP and RFP Indexes across the FSL and MS datasets.

Observations from the Table:

- Reduced Element Count in RFP Index: The RFP Index consistently exhibits a lower element count than the MFP Index. This reduction is due to the nature of RFPs, where a single RFP represents all similar segments, thus aggregating multiple MFPs.
- Memory Size Efficiency: The memory size required for the RFP Index is significantly lower than that of the MFP Index, often around 50% lower. This efficiency is attributed to the smaller number of RFPs and the reduced size of RFPs (8 bytes) compared to MFPs (20 bytes).
- Impact of Segment Size: The table demonstrates how changes in segment size affect the memory requirements of both indexes. It is observed that as the segment

size increases, the number of segments needed to identify backup data decreases, requiring less memory to store them.

Summary: Integrating the RFP Index in PR-SCAIL introduces an additional memory overhead, which is minimal and justified by the benefits in chunk-level client-side deduplication efficiency. PR-SCAIL thus significantly reduces RSD upload volume in large-scale data backups with high efficiency, with only a modest increase in memory requirements.

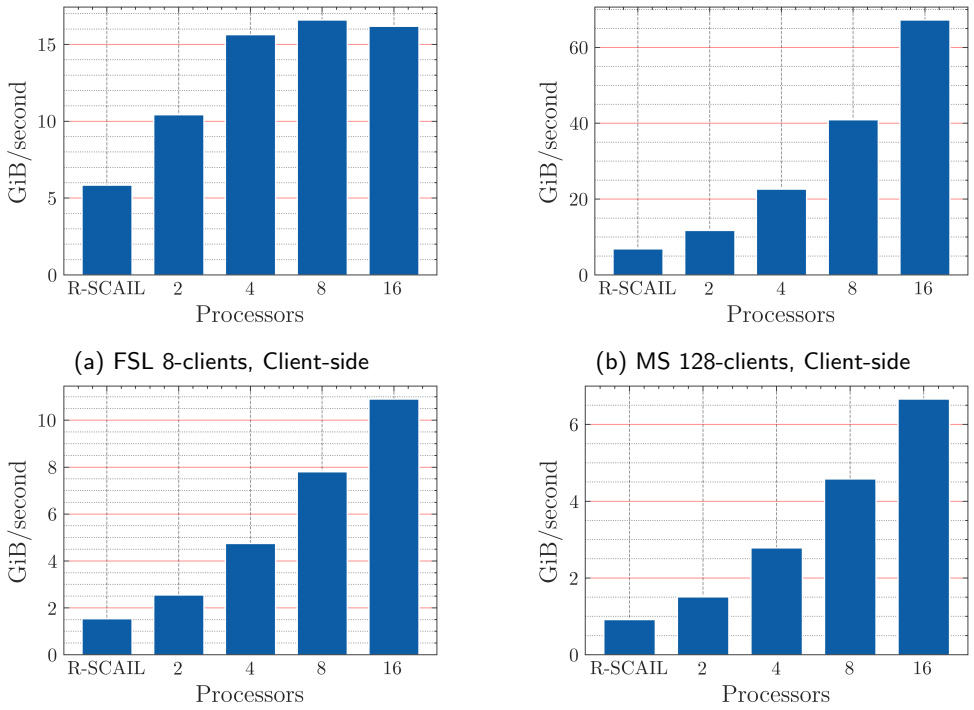
5.10.5 PR-SCAIL Throughput Analysis

This subsection analyses the throughput performance of the PR-SCAIL system, and aims to provide insights into the behaviour of PR-SCAIL under varying computational loads and client interactions. We conduct our study through experiments exploring the system's response to processor count and client volume changes.

The experimental infrastructure for this study mirrors that of P-SCAIL in Subsection 4.7.1, employing the Ray distributed framework for controlling CPU availability based on test requirements. As a shared resource across CPUs, the in-memory Redis database facilitates access to the "previously stored" and "cross-client" duplicate tables. Additionally, to ensure data integrity in container allocations, an atomic update mechanism within Redis is utilised to prevent lost updates arising from race conditions.

PR-SCAIL Multiprocessor Throughput. Our multiprocessor evaluation comprises two separate experiments. In the first experiment, we repeatedly doubled the number of processors, holding the number of clients steady. In the second, we hold the processor count at 16 and repeatedly double the number of clients. Since the FSL dataset has only eight clients, we don't include it in the second experiment.

Experiment I. We assessed the effect of increasing the number of available processors using our two datasets: the long-term FSL dataset with eight clients over 115



(a) FSL 8-clients, Client-side

(b) MS 128-clients, Client-side

(c) FSL 8-clients, Server-side

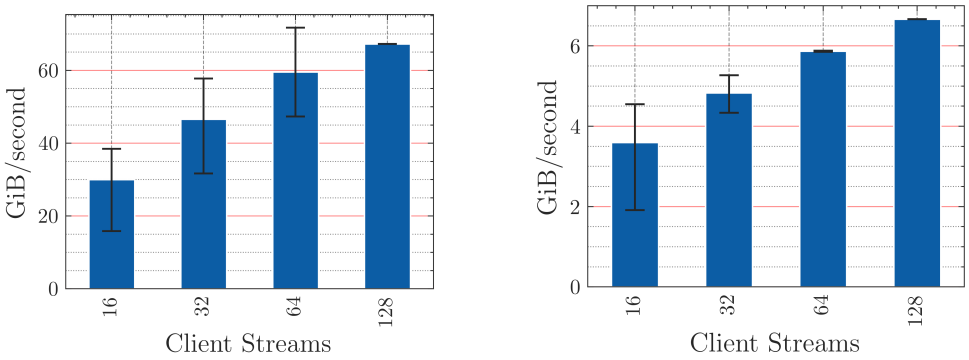
(d) MS 128-clients, Server-side

Figure 5.4: Throughput measurements in GiB/second for Client-side (5.4a, 5.4b) and Server-side (5.4c, 5.4d) deduplication on the FSL and MS datasets.

backups, and the high-volume MS dataset with 128 clients across eight backups. We benchmarked PR-SCAIL, progressively doubling the processors from 1 to 16.

The upper two bar charts of Figure 5.4 show client-side deduplication throughput for the FSL and the MS datasets. Deduplication throughput is the total size in GiB of all client's data in all backups of the dataset, divided by the time taken to deduplicate the data. Also, in Figure 5.4a, it can be seen that so there's no throughput increase beyond eight processors since there are only 8 clients. In contrast, for the MS dataset with 128 clients, throughput increases from 30.8 GiB/second to 68.0 GiB/second.

The lower two bar charts of Figure 5.4 illustrate the server-side deduplication throughput for PR-SCAIL. Throughput starts out as 1.8 GiB/second for FSL and 0.9 GiB/second for the MS dataset. With each doubling of processors, throughput improves by an average of 55% and 65%, culminating in 10.5 GiB/second (FSL) and 6.8 GiB/second (MS).



(a) MS 16-procs, Client-side (b) MS 16-procs, Server-side

Figure 5.5: Client-side and Server-side Median Deduplication Throughput with 16 processes on the MS dataset, as well as the slowest and fastest run represented as Range values.

For the FSL dataset, similar to P-SCAIL, throughput continues to rise even when the number of processors surpasses the eight clients. This reinforces that SCI's data parallelism operations further enhance throughput as more processors become available.

Experiment II. In our second experiment, we focus on accurately representing the central tendency of throughput in varying client load scenarios within the MS dataset, encompassing the first 128 clients. Once again, we measure throughput as the total number of GiBs of the specified number in 128 clients' data, divided by the time taken to deduplicate them. We adopt this approach by measuring throughput across all ordered subsets for each client group size. For instance, when evaluating 16-client loads, we don't just consider the first 16 clients but also all subsequent groups of 16. This will result in eight runs of 16 clients, constituting the entire set of 128 clients.

We then use the median of these throughput measurements as our primary statistical metric, rather than the mean. This decision is driven by the nature of our data, which is subject to variability and potential outliers. The median, being the middle value in a sorted list of numbers, more effectively represents the typical throughput in our dataset, especially in scenarios where a single client's performance could disproportionately influence the mean. This is particularly pertinent when dealing with smaller subsets of clients, where the impact of outliers is more pronounced.

Furthermore, we display the minimum and maximum throughput values as error indicators on the bar chart to provide a view of the data's variability in Figure 5.5. This approach highlights the central tendency of our throughput performance. Even though we processed the largest streams first, as outlined in Cheng and Kahlbacher [17], similar to P-SCAIL, throughput unexpectedly increased as client volumes doubled. We expected throughput to be reduced as more clients and backup volume were introduced. The throughput decrease for backups with smaller numbers of clients was caused by a few clients with large volume streams that generated substantially more new data at each backup than the other backup streams. The extended client-side duplicate lookup and server-side container allocation processing times for these larger client streams left up to 15 processors idle. However, as the client count approached 128, the combined processing duration of the smaller clients began to match or exceed that of the one or two larger client backups, leading to a more balanced workload and subsequent improvements in throughput.

5.11 Summary

In this chapter, we explored a resemblance-based strategy to reduce the volume of Redundant Segment Data (RSD) uploads, an issue intrinsic in the design of P-SCAIL. We found that PR-SCAIL's architecture lends itself to a remarkably efficient application of these techniques, compared to RMD. Leveraging the hybrid, two-phase deduplication approach of the SCAIL family, we combined intra-client exact segment-level and near-exact chunk-level client-side deduplication with cross-client chunk-level server-side deduplication. This allowed us to achieve memory efficiency through resemblance methods while maintaining high deduplication compression.

Chapter 6

Detailed Comparison

6.1 Introduction

In Chapter 3, we introduced a generic encrypted deduplication scheme, Base as well as our improved scheme SCAIL. We then sped up throughput and memory efficiency with P-SCAIL in Chapter 4. In Chapter 5 with PR-SCAIL, we added segment resemblance techniques to provide a low-memory footprint chunk-level to reduce Redundant Segment Data (RSD) volume, but this also reduced client-side deduplication throughput compared to P-SCAIL. In this chapter, we contrast these schemes across various metrics and identify some workloads and environments each scheme is most suited to.

In the next section, we'll briefly review the three algorithms, Base, P-SCAIL and PR-SCAIL and the two datasets, FSL and MS. After that are sections comparing Memory Requirements, Server Storage Requirements, Upload Volume, Single-Processor Throughput, Multiprocessor Throughput, and Cost Comparison. Finally, in Section 6.9, we'll discuss workload suitability, system trade-offs, and the preferred deployment scenarios for each system.

6.2 Algorithm and Dataset Recap

We briefly recap the Base, P-SCAIL and PR-SCAIL algorithms here.

Base Algorithm: The Base algorithm employs a memory-based chunk index that maintains ownership information, enabling it to perform client-side deduplication within individual client scopes. Like all the schemes in this paper, it implements a strict privacy protocol to protect against side-channel attacks, acknowledging only those chunks previously uploaded by the active client, thereby denying the existence of uploaded data by other clients. On the server side, it performs cross-user chunk-level deduplication using the chunk index, achieving exact duplication across users without compromising the confidentiality of upload status. After all of the chunks have been committed to containers, the chunk index is updated with the new entries, or existing entries are updated to reflect the ownership of chunks in the upload.

P-SCAIL Metadata Deduplication Algorithm: P-SCAIL stores and deduplicates metachunks as well as data chunks. However, its coarser, segment-level fingerprint deduplication leads to the upload of Redundant Segment Data (RSD). Fortunately, chunk-level server-side deduplication using Sorted Chunk Indexing (SCI) efficiently eliminates all RSD before incorporating new chunks into the server's unique data store. After all chunks have been committed to containers the chunks index is updated with new entries. No ownership information must be stored at the chunk level; P-SCAIL stores this at the segment level.

PR-SCAIL Resemblance Deduplication Algorithm: PR-SCAIL retains P-SCAIL's segment-level deduplication and adds chunk-level resemblance-based client-side deduplication. This identifies segments similar to those previously uploaded by a client, loads the CFPs of their File Recipes for those similar segments, and performs chunk-level client-side deduplication. The cross-user, chunk-level server-side deduplication using Sorted Chunk Indexing (SCI) is unchanged from P-SCAIL.

The FSL and MS datasets: Our two datasets differ significantly in the challenges they pose. The FSL dataset is the backup of 8 home directories from a small research lab, encompassing 115 backups, taken every few days over nine months. This results in 56.2 TiB of backup data, made up of 8 KiB chunks, which can be deduplicated to 431.9 GiB. This workload averages about four GiB of new data per backup day with only a minimal amount of identical chunks stored by more than one lab home directory. Throughout the backup timeline, the metadata volume increases steadily, approaching the total size of unique data stored.

The MS dataset encompasses eight weeks of complete hard drive backups from up to 140 workstations in a corporate setting, with all backups for each week grouped into eight backup batches. This large volume short-term dataset of backups consists of 45.6 TiB of data before deduplication and 2.7 TiB after deduplication. The MS dataset exhibits over 40% cross-user data in the initial backup, which likely represents operating systems or other files which are duplicated across workstations. A large average volume (350 GiB) of new data is added at each aggregated backup compared to the FSL dataset. Also, the metadata grows to 437 GiB, a sixth of the size of the data itself, after eight backups.

6.3 Memory Requirements

The memory requirements of a deduplication algorithm significantly influence both its performance and the associated hardware costs of its implementation. This is particularly true in the context of differing workloads, where varying data characteristics can lead to distinct memory demands. Multi-stage algorithms further accentuate this complexity, as each component may have unique memory requirements. Understanding these differences is important for optimizing algorithm design and implementation.

In this section, we delve into a detailed breakdown of the total memory requirements by component for each algorithmic scheme, against the backup of the FSL and

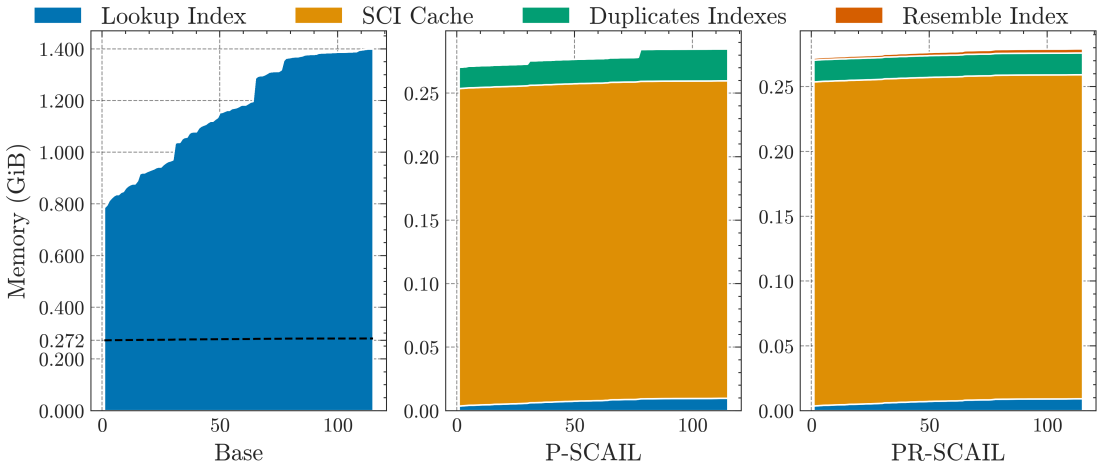


Figure 6.1: FSL Dataset: Stacked areas charts showing memory requirements by component in GiB for the Base and PR-SCAIL schemes over 115 backups. The horizontal dashed line on the Base chart is the total memory requirements for PR-SCAIL. Scales differ between charts.

MS datasets. The components of memory usage under examination are:

1. **Lookup Index** (All Systems): Used for answering client-side deduplication queries. In the Base system, the keys are CFPs of previously saved chunks, and the values include a CID and a list of owners used in privacy protection. In contrast, P-SCAIL and PR-SCAIL use an MFP key, with values also comprising a CID and ownership information. Use of MFPs enables an index orders of magnitude smaller than CFPs.

2. **SCI Cache** (P-SCAIL and PR-SCAIL): A constant-sized cache set up at system initialisation. It includes 128 MiB for holding a single SCI bin page in memory and a 128 MiB cache for new $CFP \rightarrow CID$ mappings. This cache is flushed to disk and cleared if it becomes full.

3. **Duplicates Indexes** (P-SCAIL and PR-SCAIL): During SCI on the server, transient indexes, comprising 'previously saved' and 'inter-user' indexes, are generated anew for each backup. The 'previously saved' index holds mappings of $CFP \rightarrow CID$ for chunks uploaded from a client that has been previously saved,

the 'inter-user' index stores CFPs for chunks uploaded by more than one client in a backup batch. These indexes, generated during SCI processing, facilitate exact deduplication in multiprocessor scenarios.

4. **Resemble Index** (PR-SCAIL only): A specialised, compact index that uses RFPs as keys. The value associated with a key is a set of CIDs, which point to metadata containers which hold previously saved similar segments containing the RFP key.

6.3.1 FSL Dataset Memory Requirements

The charts in Figure 6.1 show the memory requirements for the Base, P-SCAIL and PR-SCAIL systems over the course of backing up the FSL dataset. Be aware that the same scale is not used for all charts. This enables more detail to be shown for the different memory usage components. Also, for reference, the horizontal dashed line in the Base chart reflects the total memory requirements for P-SCAIL and PR-SCAIL compared to Base.

Upon completing the 115 backups, the Base scheme consumes a total of 1.4 GiB of memory. In contrast, the SCAIL family of algorithms — P-SCAIL and PR-SCAIL — showcases remarkable efficiency, requiring merely 285 MiB of memory, approximately one-fifth of the total memory utilised by Base. Most of this memory usage for both P-SCAIL and PR-SCAIL is in the 256 MiB SCI Cache. An additional 9 MiB is occupied by the MFP index. For P-SCAIL, the Duplicates Indexes require 25.7 MiB, while in PR-SCAIL, these indexes are further optimised to use only 17.2 MiB. In PR-SCAIL, the Resemblance Index adds a minimal 3 MiB to the overall memory footprint, underscoring the efficient memory management inherent in the SCAIL algorithms.

The Lookup Index in both schemes grows as new (previously unseen) data comes into the system. The CFP index in Base grows on average by 0.68% per backup, while the MFP index actually grows faster than the CFP index at 1.45% per backup. This

faster growth is due to the modification of previously saved segments. While the Base Lookup Index grows only with unique CFPs, a new MFP is generated whenever a segment is modified, even if only a single chunk is new. However, the MFP Index starts very small, and even with its faster growth rate, it ends up being less than 9 MiB, which is around $\frac{1}{150}$ the size of the Base Index.

If we extrapolated out the higher growth rate of MFPs to CFPs out to 1 PB of unique data (assuming the same relative growth rate), the SCAIL Lookup Index would actually have to represent 3.12 PB of data. But this still would require less than 50 GiB of memory, so would be feasible with today's servers.

The bottom line is that for the FSL dataset, the SCAIL algorithms use much less memory and, importantly for scaling operations, in absolute size, grow only a modest amount with each backup.

6.3.2 MS Dataset Memory Requirements

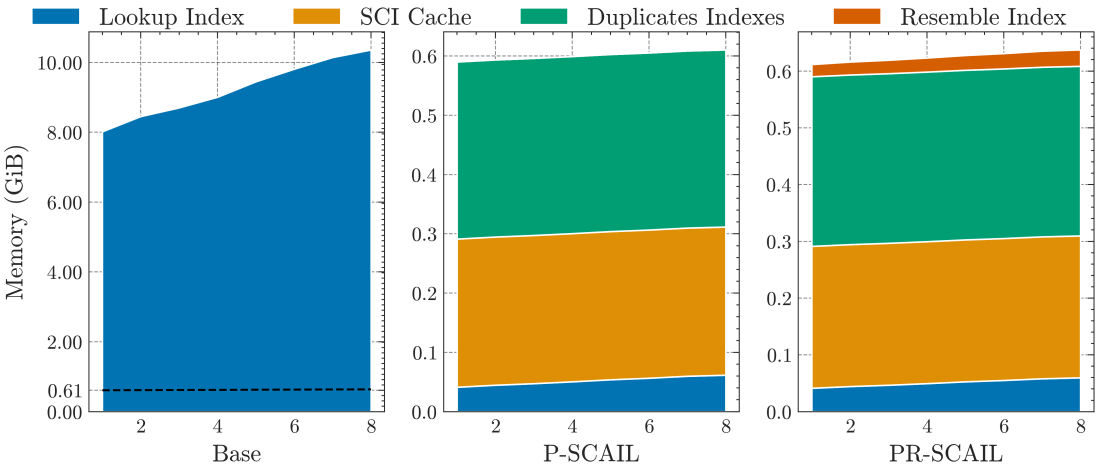


Figure 6.2: MS Dataset: Stacked area charts of memory requirements (GiB) by component across eight backups for the Base and PR-SCAIL schemes. The dashed line in the Base chart shows the total memory requirement for PR-SCAIL. Scales differ between charts.

The charts in Figure 6.2 illustrate the memory requirements for the MS dataset. The scale of the P-SCAIL and PR-SCAIL charts is smaller to detail the components and still

assess their growth rate. The horizontal dashed line on the Base chart represents the comparative total memory requirements of P-SCAIL and PR-SCAIL against Base.

Upon completing the eight weekly backups, the Base system consumes 10.4 GiB of memory. In contrast, the SCAIL-based algorithms require only about a total of 650 MiB, approximately $\frac{1}{16}$ of Base's memory usage. This considerable reduction in memory requirements demonstrates the efficiency of the SCAIL approach. The majority of memory in P-SCAIL and PR-SCAIL is allocated to the 256 MiB SCI Cache (identical with the FSL dataset backup) and the 305 MiB Duplicates Indexes. The Duplicates Indexes in the MS dataset require more memory than in the FSL dataset due to the substantial amount of cross-user duplicate data in MS's initial backup. For the rest of the indexes, the MFP Index utilises 61 MiB, while the Resemblance Index uses about 29 MiB.

The MS dataset also reveals differences in the average growth rate of the Lookup Index. The Base's Lookup Index grows by an average of 4.2% per backup, whereas the MFP index in the SCAIL algorithms grows faster, at around 7.0% per backup due to modified segments. Despite its faster growth, the MFP Index starts from a much smaller base and remains under 60 MiB, less than $\frac{1}{160}$ the size of the Base Index.

If we extrapolated out the higher growth rate of MFPs to CFPs out to 1 PB of unique data (assuming the same relative growth rate), the SCAIL Lookup Index would actually have to represent around 3 PB of data. But this still would require less than 50 GiB of memory, so would be feasible with today's servers.

6.3.3 Summary of Memory Requirements Findings

The analysis of memory requirements across both the FSL and MS datasets highlights the efficiency of the SCAIL family of algorithms. These algorithms demonstrate substantial memory savings, utilizing merely one-fifth and one-sixteenth of the memory required by the Base scheme for the FSL and MS datasets, respectively. Such findings

emphasise the scalability of the SCAIL deduplication family and suggest that systems managing larger volumes of data can achieve even greater comparative memory advantages.

While memory efficiency is a critical factor in the scalability of encrypted deduplication systems, it is not the sole determinant of their effectiveness. With the memory requirements comprehensively analysed, our attention now shifts towards evaluating the server storage requirements of each deduplication scheme, a crucial aspect in understanding the cost and performance of these systems.

6.4 Server Storage Requirements

The fundamental objective of encrypted deduplication systems lies in their ability to significantly amplify the volume of data that can be backed up on a server relative to the actual volume of data stored. This is particularly crucial in long-term backup scenarios, exemplified by the FSL dataset, where the challenge often lies in the continuous growth of metadata in proportion to the logical data size presented to the system. By modifying the metadata deduplication technique pioneered by Metadedup [44], the SCAIL-family of algorithms substantially reduces server storage requirements. This approach optimises storage utilisation and enhances the efficiency and scalability of encrypted deduplication systems, making them more able to handle extensive data backups with high throughput over prolonged periods.

The storage requirements on the server can broadly be grouped into storing unique chunk data and metadata used for Restore operations. We show how data and metadata grow over backup generations for each dataset.

Since all systems under consideration are exact deduplication systems, we also present charts that remove the data storage metric and provide greater detail into each system's types and volume of metadata.

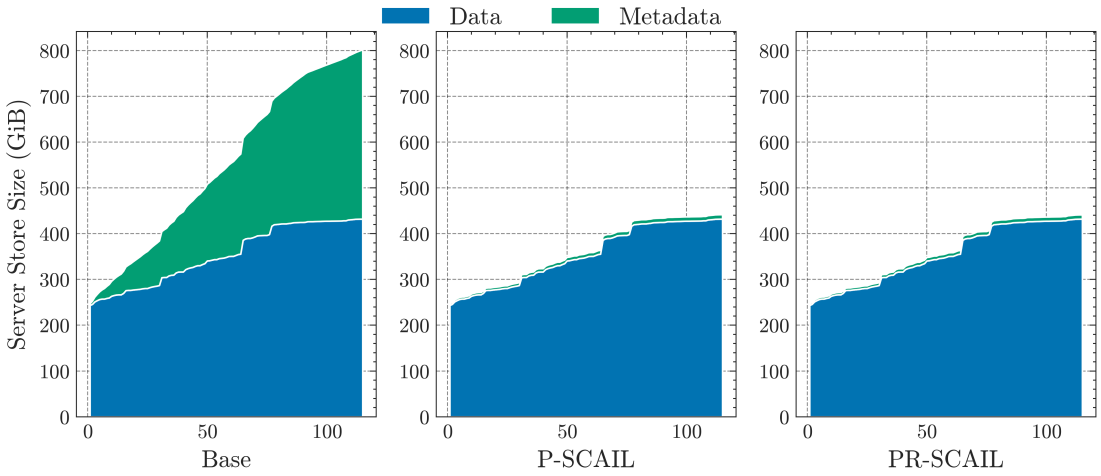


Figure 6.3: FSL Dataset: Stacked areas charts showing cumulative data and metadata storage volume on the server for Base and PR-SCAIL with 8 KiB chunks and 2 MiB metachunks.

6.4.1 FSL Server Storage Requirements

All schemes store the same 431.9 GiB volume of unique server data, as seen in Figure 6.3. The Base approach adds 369.9 GiB of metadata, while P-SCAIL and PR-SCAIL deduplicate File and Key Recipes, so only store 9.8 and 9.5 GiB of metadata, respectively. In total, Base must store 801 GiB and the SCAIL-based schemes only 442 GiB, which is 44% smaller.

In Figure 6.4, we take a closer look at the metadata storage requirements. The metrics charted in the metadata detail charts (note that scales differ between charts) are:

1. **Lookup Index** (all systems, **visible only in MS dataset, which appears in the next subsection**): This is the sum of the CFP Index for Base, MFP Index for P-SCAIL and PR-SCAIL, and Resemble (RFP) Index for PR-SCAIL.

2. **SCI Bins** (P-SCAIL and PR-SCAIL): This is the full chunk index used for server-side deduplication, stored on disk.

3. **Recipes** (all systems): These are File and Key Recipes used for decryption and

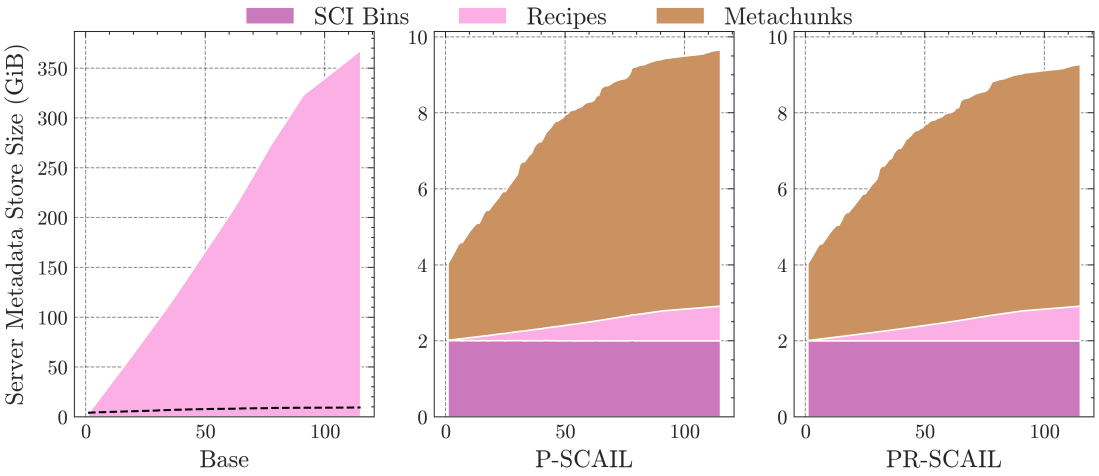


Figure 6.4: FSL Dataset: Stacked area chart showing accumulated metadata storage volume by component. Scales vary across charts. The dashed line in the Base chart shows P-SCAIL/PR-SCAIL's accumulated volume for comparison. Lookup Index storage is omitted due to its small size.

Restore operations. Base uses CFPs, SCAIL algorithms use MFPs.

4. **Metachunks** (P-SCAIL and PR-SCAIL): These are the encrypted metachunks referenced by SCAIL-based File and Key Recipes.

The server will need to save disk-based copies of all memory resident indexes, but for the FSL dataset, they are too small to be visible on these charts at 1.4 GiB for Base and less than 14 MiB for the SCAIL schemes.

The Base scheme stores 368.5 GiB of File and Key Recipes for Restore operations. The SCAIL schemes store 936.7 MiB of File and Key Recipes, but must also store 6.75 GiB of deduplicated Metachunks. For server-side SCI deduplication, 2 GiB is required for the SCI Bins, constituting the chunk-level index for the SCAIL schemes. In total, Base stores 369.9 GiB of metadata, P-SCAIL stores 9.7 GiB, and PR-SCAIL stores 9.3 GiB, which is a savings of 97.4% for the Base metadata storage requirements.

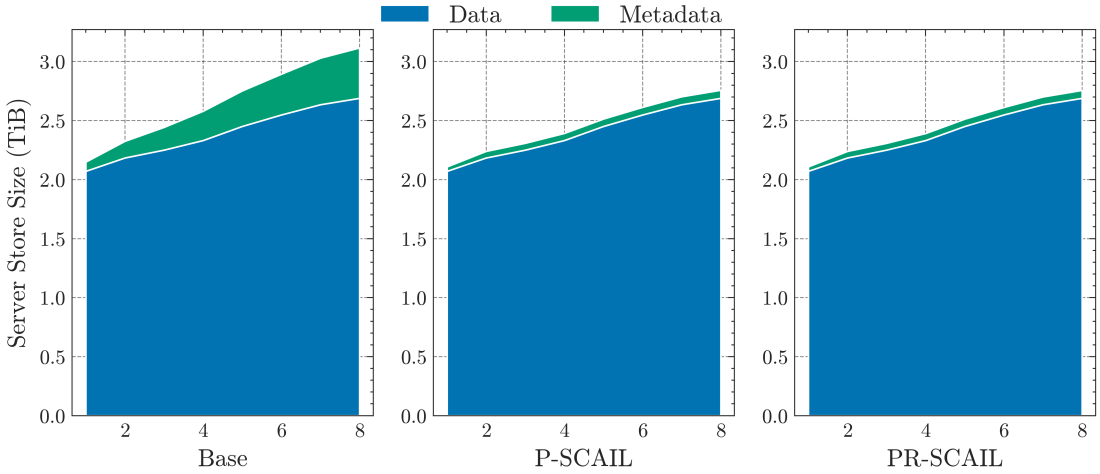


Figure 6.5: MS Dataset: Stacked areas chart showing sccumulated server storage volume by component for the Base, P-SCAIL and PR-SCAIL schemes.

6.4.2 MS Server Storage Requirements

All schemes store the exact same 2.7 TiB volume of unique data on the server as shown in Figure 6.5. Base must also store 437.1 GiB of metadata for a total of 3.1 TiB, while P-SCAIL and PR-SCAIL must store around 70 GiB of metadata, for a total of 2.8 TiB, for a savings of 11.5%.

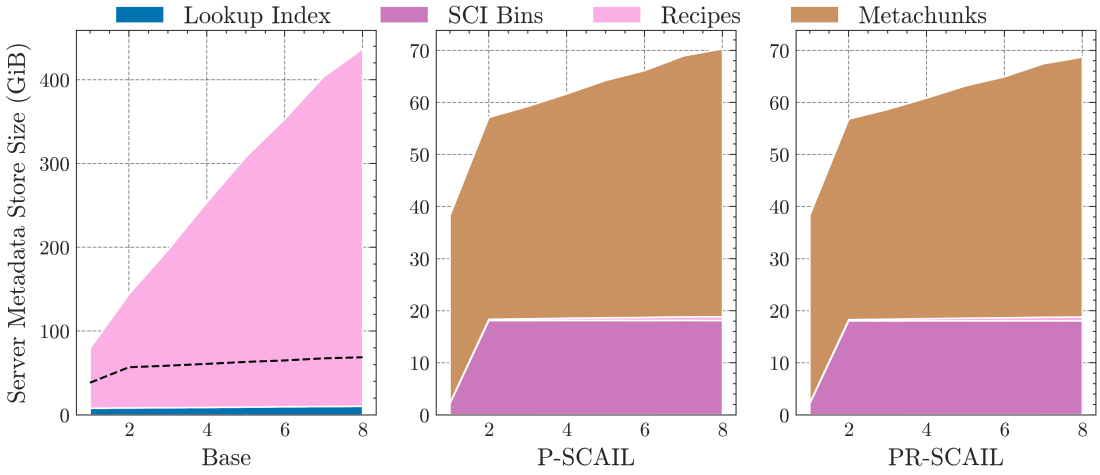


Figure 6.6: MS Dataset: Stacked areas chart showing accumulated metadata server storage volume by component. The dashed line in the Base graph indicates the volume of metadata storage required for the SCAIL schemes. Not all graphs use the same scale.

In the metadata detail charts at Figure 6.6 (note that not all chart scales are the same), Base's 10.4 GiB Lookup Index combined with File and Key Recipes of 426.7 GiB takes a total of 437.1 GiB of storage. In the P-SCAIL and PR-SCAIL charts, the Lookup Index and the Resemblence Index are included but not distinctly visible, since they are less than 100 MiB total. The SCI Bins for the SCAIL algorithms required 18.1 GiB. This is almost twice the size of Base's Lookup Index, since we preallocate padding in these bin files for performance reasons (see Subsection 3.5). The File and Key Recipe storage is a mere 754 MiB, and the volume of deduplicated Metachunks referenced by the recipes is 51 GiB. P-SCAIL and PR-SCAIL reduce the metadata storage required by the system from Base's 437.1 GiB to 70.3 GiB, an 83.9% reduction.

6.4.3 Summary of Storage Findings

In summary, the SCAIL algorithms perform client-side deduplication at the coarse segment level (e.g. 2 MiB chunks), but for server storage, are able to take advantage of the fine, chunk level (e.g. 8 KiB chunks) during server-side, cross-user deduplication. When combined with metachunk deduplication, it enables SCAIL algorithms to reduce metadata storage requirements by 97.4% (FSL dataset) and 83.9% (MS dataset).

While the SCAIL family of algorithms dramatically reduces metadata storage requirements across the board, upload volume results are varied. In some cases, the SCAIL overall upload can exceed the Base volume. In the next section, we'll take a detailed look at the upload volume.

6.5 Upload Volume

Client-side deduplication is used to reduce upload volume in client-server-based backup systems. It's based on the observation that data that has been uploaded to the server doesn't have to be uploaded again. Our investigation also requires that client data be protected from side-channel attacks (see Subsection 3.9). The system should avoid

disclosing whether another client has uploaded a piece of data. So, for a given client, we limit client-side deduplication processing to data that *only this client* has previously uploaded. Also, there will be cases where multiple clients will submit identical data to the server within the same backup cycle. While the server could detect this scenario and choose one client to upload the data, this would also violate data privacy, so the system requires all the clients to upload identical data within a backup cycle.

Quantifying the cost implications of upload volumes proved challenging. However, by estimating upload times, we discovered that the excess uploads of RSD associated with the P-SCAIL algorithm had a negligible impact, as outlined in Section 3.8. The extra time required for P-SCAIL to handle its increased upload volume is an average of less than a minute per backup for each client, underscoring its minimal effect on overall performance.

This section examines and compares the upload volumes of the three schemes: Base, P-SCAIL and PR-SCAIL, presenting results against the two distinct datasets, FSL and MS. While the bulk of the upload volume is unique encrypted data chunks, other data like metadata and chunks that are redundant segment data must be uploaded by some algorithms to the server as well. Understanding the components contributing to the total upload volume helps assess each deduplication scheme's overall impact and efficiency.

6.5.1 FSL Upload Volume

Upload Volume By Components

In the upload volume chart for Stage 3 shown in Figure 6.7, the accumulating upload component values are made up of:

1. **Data** (all schemes): This is the volume of all unique (deduplicated), encrypted chunk data uploaded to the server.

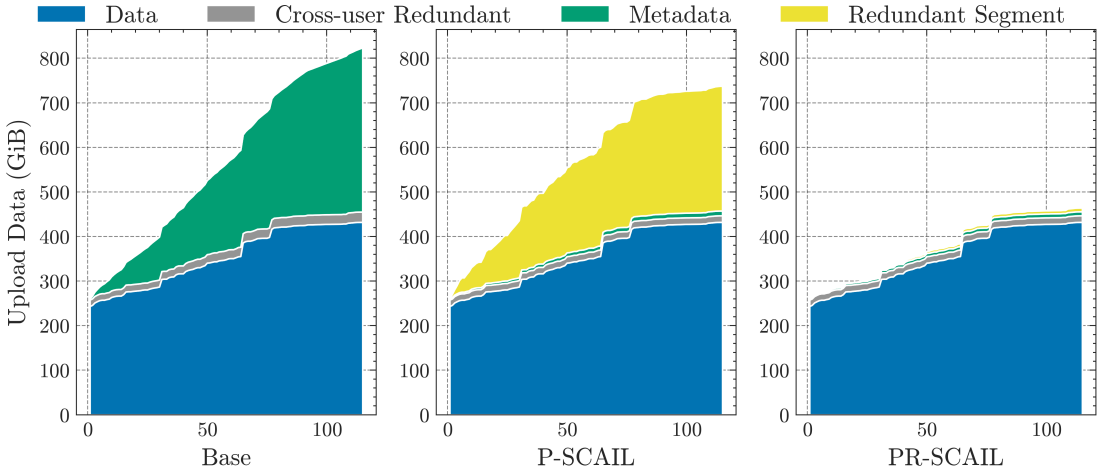


Figure 6.7: FSL Dataset: Stacked area chart showing accumulated upload volume by component for the Base, P-SCAIL and PR-SCAIL schemes.

2. **Cross-user Redundant** (all schemes): When multiple clients upload new, identical data, the 2nd and subsequent copies will fall into this category.

3. **Metadata** (all schemes): This includes all upload volume other than non-data upload volume (recipes, sorted CFPs, metachunks) each system must upload.

4. **Redundant Segment** (P-SCAIL and PR-SCAIL only): The volume of RSD; the encrypted chunk data in each client's upload that they have saved in a previous backup. RSD is always discarded on the server after cross-user deduplication.

The comparative chart for the FSL dataset reveals distinct patterns in upload volume across the Base, P-SCAIL, and PR-SCAIL schemes based upon 8 KiB chunks and 2 MiB segments. In the Base scheme, there is a near-equal distribution between Data and Metadata upload volume over the course of 115 backups, with a small amount of Cross-user Redundant data, primarily in the initial backup. This pattern reflects Metadata's outsized influence on the upload volume of long-running backups, as it constitutes nearly as much upload volume as the data itself.

In contrast, the P-SCAIL scheme exhibits the same volume of Data uploads as Base, but the Metadata component is dramatically reduced – by approximately 97%. This

results in a substantial server storage reduction for long-running backups, see Subsection 6.4. However, this reduction in upload volume is counterbalanced by a notable increase in RSD uploads, nearly matching the volume saved in Metadata uploads. This reflects the segment-level interaction of P-SCAIL clients with the server and highlights the trade-off between Metadata reduction and the addition of RSD. Fortunately, all RSD is discarded on the server after cross-user deduplication.

PR-SCAIL uploads the same volume of Data and Metadata as P-SCAIL, but manages to substantially lower RSD – by about 96%. This reduction demonstrates the efficiency of PR-SCAIL's segment resemblance techniques in minimizing RSD. However, it's important to note that the client-side deduplication process in PR-SCAIL is slower, although still much faster than Base, which we will explore in detail in the subsequent throughput analysis sections.

Overall, Base uploads a total of 823.5 GiB, P-SCAIL uploads 737.2 GiB (11% less than Base), and PR-SCAIL uploads 464.0 GiB (77% less than Base).

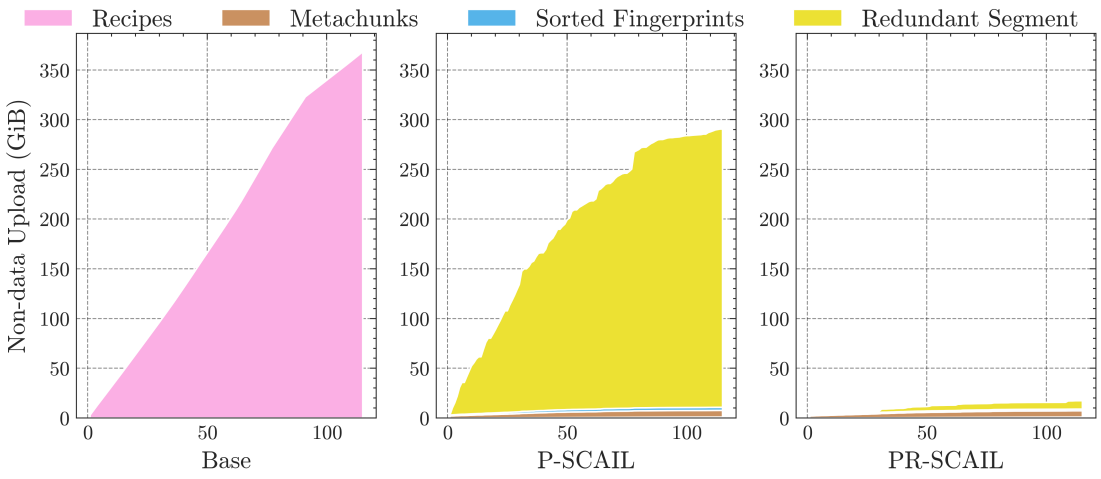


Figure 6.8: FSL Dataset: Stacked area chart showing accumulated upload volume by component, after removing cross-user and chunk data upload from Figure 6.7.

Non-data Upload Component Volume

Since all schemes upload the identical volume of Data and Cross-user Redundant data, we drop these metrics and show a 'non-data' upload volume chart in Figure 6.8 to contrast the other required upload volumes. The data values in the 'non-data' chart are:

1. **Recipes** (all schemes): File and Key Recipes required for Restore operations.
2. **Metachunks** (P-SCAIL and PR-SCAIL only): These are the deduplicated encrypted metachunks which are referenced in the SCAIL family for File and Key Recipes used for Restore.
3. **Sorted Fingerprints** (P-SCAIL and PR-SCAIL only): Each client uploads this list for use in the chunk-level SCI algorithm, where server-side chunk-level deduplication is performed.
4. **Redundant Segment** (P-SCAIL and PR-SCAIL only): The summed volume of encrypted chunk data in each client's upload that they have previously saved.

In the 'non-data' chart analysis, we observe a reduction in the combined volume of File and Key Recipes and Metachunks for P-SCAIL and PR-SCAIL compared to the volume of File and Key Recipes in Base. However, the introduction of a significant volume of RSD by P-SCAIL is apparent, albeit still contributing to a reduced overall upload volume when compared to Base. This reduction gives the total non-data upload volume for P-SCAIL as 290.7 GiB, a 21% decrease from Base's 368.5 GiB.

Continuing the analysis of the 'non-data' charts, in the PR-SCAIL scheme, the RSD is considerably diminished. The total non-data upload volume for PR-SCAIL stands at 17.4 GiB, which is 95% less than the Base scheme. This highlights the effectiveness of PR-SCAIL in minimizing the Metadata File and Key Recipe uploads *and* the reduction of RSD, demonstrating its superior efficiency in reducing upload volumes. Once again,

we note the caveat that PR-SCAIL is slower at client-side deduplication than P-SCAIL, which we detail in the subsequent throughput analysis section.

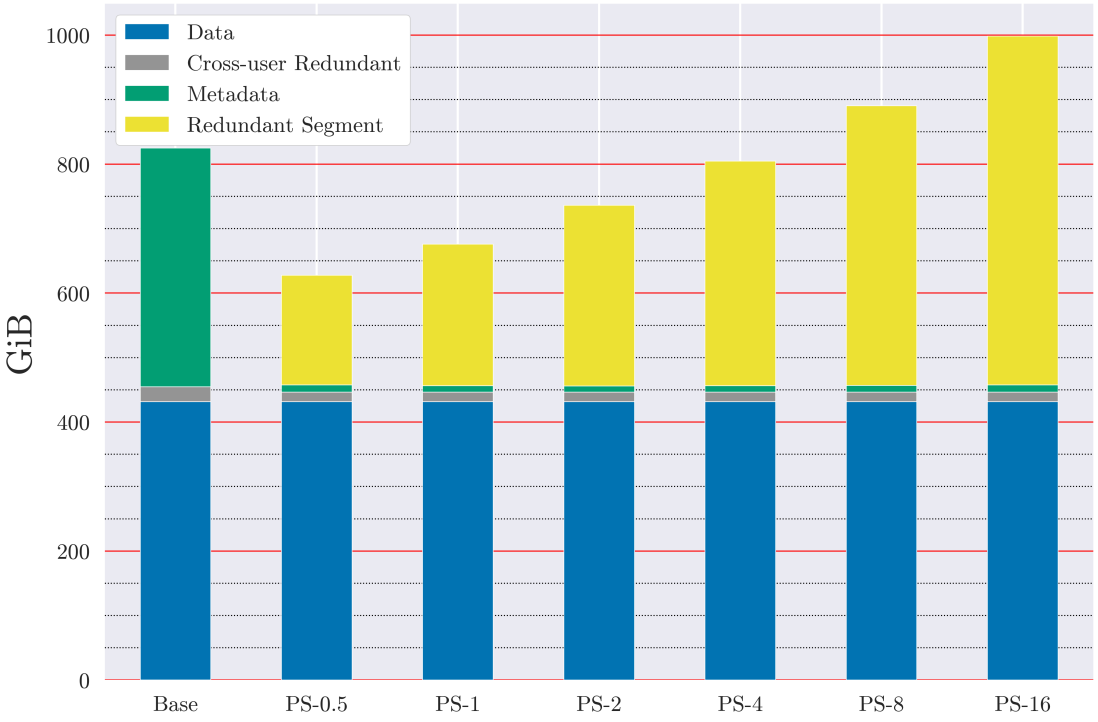


Figure 6.9: FSL Dataset: Total upload volume by component. The Base scheme is the first bar, subsequent bars labeled PS-x indicate a P-SCAIL scheme with segment size x in MiB.

Segment Size Effect on Upload Volume

The analysis above showed accumulating upload volume sums for the Base scheme and the SCAIL schemes with 2 MiB segments. Here, we vary the segment size for the P-SCAIL scheme and detail its effect on the total upload volume for each component type, as well as compare the total upload volume. The results are shown in Figure 6.9.

The Base scheme's upload volume, shown as the first stacked bar in the chart, serves as our benchmark, reflecting the optimal upload volume for a traditional MLE backup system. It is characterised by a nearly equal distribution between metadata and data volumes after the 115 backups.

The subsequent P-SCAIL schemes in the chart exhibit a significant reduction in

metadata volume. An incremental increase in RSD offsets this reduction as segment size varies from 0.5 MiB to 16 MiB. This shift underscores the algorithm's efficiency in metadata management while highlighting the trade-off with RSD. Initially, for a segment size of 0.5 MiB, the RSD constitutes less than half of the metadata volume observed in the base scheme. However, with increasing segment size, the volume of uploaded metadata by P-SCAIL approaches that of Base. For segment sizes up to 4 MiB, P-SCAIL uploads less metadata than Base; however, for segment sizes greater than 4 MiB, P-SCAIL exceeds Base in uploaded metadata volume.

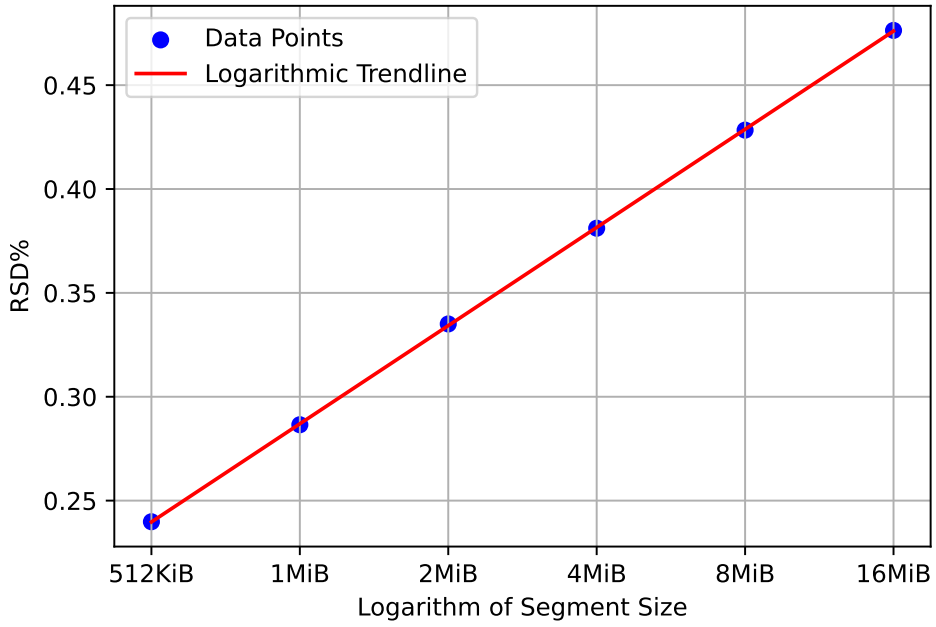


Figure 6.10: FSL Dataset: Percentage of Redundant Segment Data (RSD) of the total upload volume, as a function of segment size, showing observed (dots) and predicted (line) percentage of RSD across a range of segment sizes.

These findings highlight the relationship between segment size and upload volume composition in the P-SCAIL deduplication system. To identify the prevalence of RSD for a dataset, we introduce a metric that calculates the proportion of RSD volume to the total upload volume, then subsequently normalised this ratio by the number of backups. This approach provides a measure to gauge the impact of RSD.

For example, consider the total upload volume for 2 MiB segments, which amounts to 737.2 GiB, with 279.9 GiB (or 38%) constituting RSD. Normalised to volume per backup, RSD accounts for 0.33% of the total upload volume per backup. We applied this computation across various segment sizes, and the results are depicted in Figure 6.10.

This chart revealed a positive correlation between segment size and the percentage of RSD, which can be attributed to the fact that even altering a single chunk within a segment causes the classification of nearly the entire segment as RSD. Larger segments consequently produce more substantial RSD. We calculated a logarithmic curve to fit the empirical data, yielding the formula

$$RSD \text{ percentage} = 0.068 \times \log(\text{segment size}) + 0.287.$$

This model enables the estimation of RSD volumes for varying segment sizes. We also found that segment size did not impact the small amount of RSD that PR-SCAIL missed.

6.5.2 MS Upload Volume

As we shift our focus to the MS dataset, we encounter different dynamics in the upload volume analysis. The MS dataset, characterised by a much larger number of users and a much higher volume of deduplicated data, presents unique challenges and insights compared to the FSL dataset. Notably, with only eight backups, the MS dataset offers fewer opportunities for deduplication, thereby influencing the efficiency of each scheme differently. Additionally, the presence of a substantial amount of Cross-user Redundant data, likely due to common operating system files shared across systems, adds another layer to our comparative analysis. This section will delve into how the Base, P-SCAIL, and PR-SCAIL schemes perform under these conditions, highlighting the upload volume that emerges with this more diverse and larger but shorter-duration

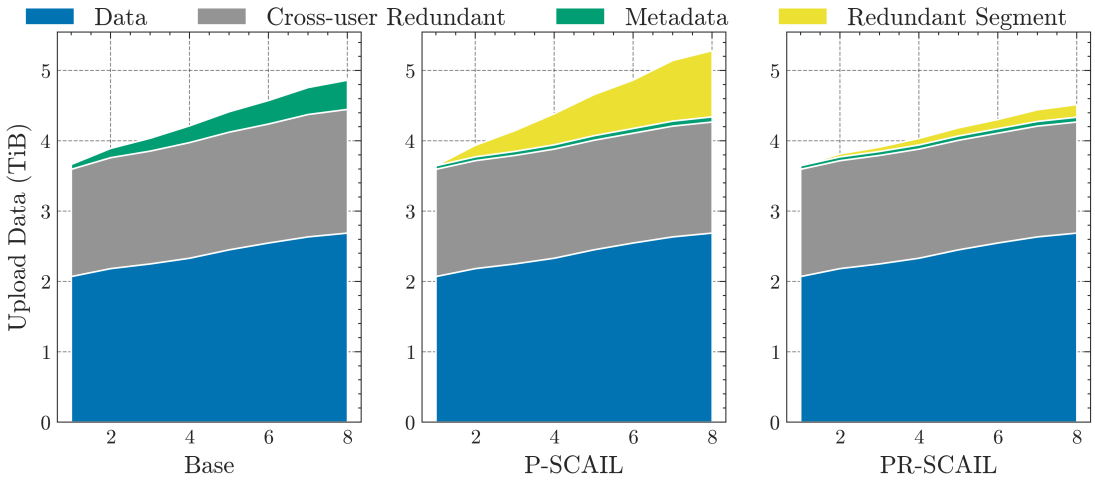


Figure 6.11: MS Dataset: Stacked area chart showing accumulated upload volume by component for Base, P-SCAIL and PR-SCAIL.

dataset.

Upload Volume By Component

As defined in the FSL dataset analysis subsection above, the chart values, such as Data, Cross-user Redundant, etc., follow the same meaning in the MS dataset comparison.

In the charts in Figure 6.11, we can see the upload volume in TiBs and that the initial backup has 40% identical cross-user data. Metadata grows modestly, but RSD grows faster in the P-SCAIL dataset, while RSD is reduced by 80% with PR-SCAIL. The total upload volume for Base is 4.9 TiB, but for P-SCAIL, it increases 8% to 5.3 TiB, while PR-SCAIL decreases in volume by 7% to 4.5 TiB.

Non-data Upload Component Volume

In the non-data upload volume chart in Figure 6.12, we see the substantial volume of RSD in the P-SCAIL scheme, and that it has been reduced 80% by PR-SCAIL. At this scale in the charts, Metachunks are visible at around 51 GiB, but File and Key Recipe volume has been reduced to 754 MiB, so it is not visible. Sorted Fingerprints start at 15

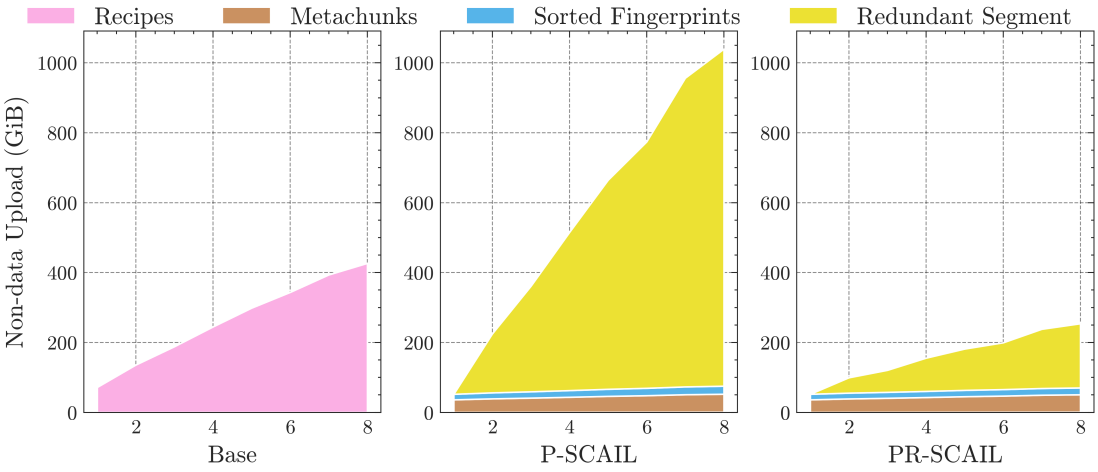


Figure 6.12: MS Dataset: Stacked areas charts for the non-data Upload Volume Chart for 2 MiB Segments.

GiB on the first backup, and don't grow much thereafter, reflecting that the volume of new data to be uploaded does not grow appreciably after the first backup.

Segment Size Effect on Upload Volume

In the previous subsection, we showed the growth of the upload volumes for the upload categories for the three schemes. Here we compare Base upload volumes to P-SCAIL upload volumes for various segment sizes. In Figure 6.13, the first stacked bar is for the Base scheme, which is our benchmark for the upload volume of a traditional, encrypted deduplication backup scheme. The Metadata volume of 437 GiB is only about 15% of the 2.7 TiB Data volume for eight backups. Recall that the FSL Dataset with 115 backups had accumulated nearly 50% Metadata. Metadata grows with the volume of data presented to the server, while Data growth reflects only new unique chunk data. As the number of backups grows, the SCAIL algorithms will increase their advantage in upload volume compared to Base. With only eight backups, the reduction in Metadata upload volume for the smallest segment size examined of 512 KiB is offset by the added volume of RSD.

Not surprisingly, as segment size increases with the P-SCAIL schemes, RSD in-

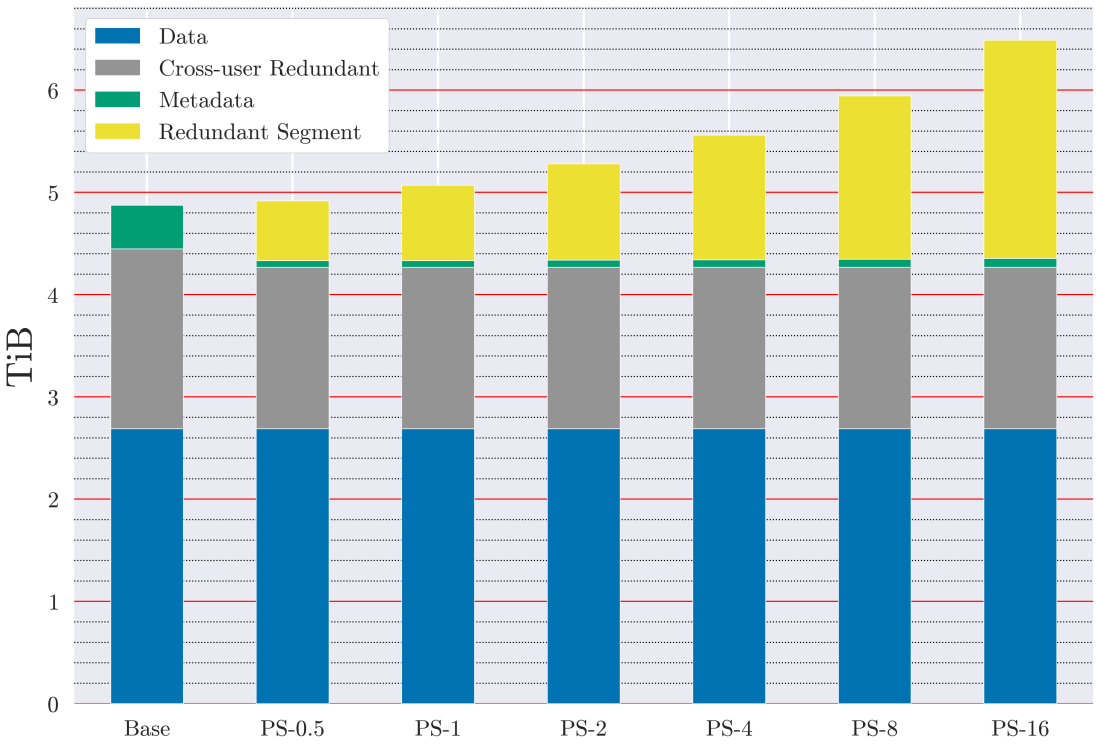


Figure 6.13: MS Dataset: Stacked bar chart showing the total upload component volumes after the Eight Backups of the MS Dataset.

creases. For the largest segment size examined of 16 MiB, the total upload volume is 34% larger than Base.

We investigate the relationship between segment size and RSD with the P-SCAIL scheme. The methodology employed mirrors the FSL dataset analysis with the MS dataset. As an example, for 2 MiB-sized segments, the total upload volume was 5.2 TiB, of which 964.0 GiB (18%) was RSD. Normalised over the eight backups gives 2.5% of upload overhead per backup for RSD. For the MS dataset, the quantified relationship between the segment size and RSD is encapsulated within the logarithmic fitting formula given by

$$RSD \text{ percentage} = 0.761 \times \log(\text{segment size}) + 1.805.$$

The analysis is visually supported by Figure 6.14, which contrasts the logarithmic

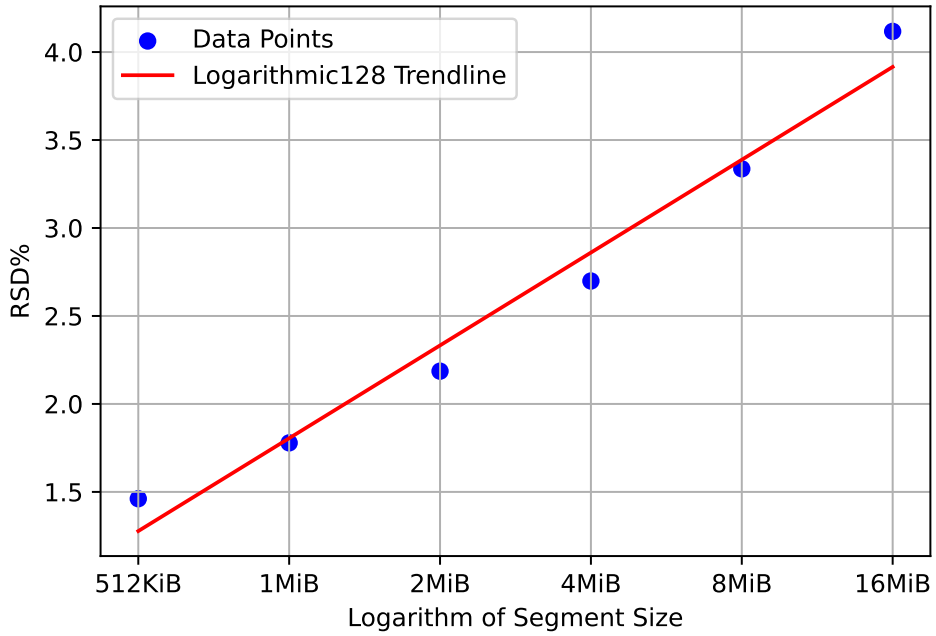


Figure 6.14: MS Dataset: Percentage of Redundant Segment Data (RSD) of the total upload volume, as a function of segment size, showing observed (dots) and predicted (line) percentage of RSD across a range of segment sizes.

trendline with the empirical data points.

6.5.3 Insights from Upload Volume Analysis

The comparative analysis of upload volumes for Base, P-SCAIL and PR-SCAIL schemes across both the FSL and MS datasets offers insights into the efficiency and scalability of the deduplication strategies. The FSL dataset, characterised by its longer-term backup scenario, showcases the substantial advantage of the SCAIL algorithms in reducing upload volume, particularly of Metadata. The Base scheme's upload volume, heavily influenced by Metadata, is significantly higher than that of P-SCAIL and PR-SCAIL, which implement Metadedup's technique to deduplicate Metadata effectively.

For total upload volume with the FSL dataset, P-SCAIL demonstrates a reduction compared to Base with segment sizes up to 4 MiB. This is primarily due to the dramatic reduction in Metadata. However, as segment sizes grow, the presence of RSD in

P-SCAIL increases and eventually offsets the saving of Metadata upload volume. The upload volume, when considered as a percentage of the total volume for each backup, RSD is actually quite small.

Because it effectively suppresses RSD in addition to metadata deduplication, the PR-SCAIL scheme yields smaller upload volumes than Base for all segment sizes. However, this strategy incurs a trade-off in the form of reduced client-side deduplication throughput compared to P-SCAIL, although it is still many times faster than Base for client-side deduplication. This trade-off highlights the efficiency of the resemblance-based deduplication approach employed by PR-SCAIL.

The MS dataset presents a different challenge with its distinct short-term but high-volume backup scenario. Also, the initial backup is characterised by a high volume of Cross-user Redundant data, which all schemes must upload due to privacy constraints. For the smallest segment size tested, 512 KiB, P-SCAIL uploads about the same volume as Base. Larger segment sizes increase the volume beyond Base, even though the additional upload amounts to, at most, 34% of the total upload volume, which equates to around 4% per backup. We anticipate that for longer backup workloads, which will accumulate metadata that is deduplicated by P-SCAIL but not by Base, P-SCAIL will demonstrate a growing advantage in upload volume reduction compared to Base.

PR-SCAIL significantly reduces RSD, along with File and Key Recipe deduplication, resulting in a lower total upload volume than Base for all segment sizes. This demonstrates its ability to efficiently manage high-volume, short-term backup scenarios, but at reduced throughput compared to P-SCAIL.

The RSD upload volume for the MS dataset, growing to nearly one TiB for the MS dataset using 2 MiB segment size, was a strong motivating factor in developing the PR-SCAIL scheme. However, this scheme, working at the chunk-level for client-side deduplication, comes with slower client-side deduplication throughput than P-SCAIL, but still many times faster than Base, as outlined in the throughput sections to follow.

6.6 Single-processor Throughput

The SCAIL suite has shown notable improvements in handling larger and longer-running dataset workloads with an increased number of concurrent clients compared to the Base scheme. A high level of throughput is essential to take advantage of this scalability in a practical implementation. In this section, we examine the single-processor throughput against our datasets with the Base, P-SCAIL, and PR-SCAIL schemes.

As detailed in Subsection 4.7.1, we calculate throughput performance for client-side and server-side deduplication by measuring the total volume of logical data presented to the server against the wall clock time for backups to reach completion in specific stages.

First, we'll examine the impact of segment size on client-side deduplication throughput. It's important to note that the charts illustrate "instantaneous" throughput to depict variations during the backup process. That is, the instantaneous throughput (in GiB/second) is the total volume of data deduplicated in a single backup generation, divided by the time taken to perform the deduplication. Additionally, our description and comparative analysis of performance utilises the 'cumulative average'. We define cumulative average as the total number of GiB of data deduplicated for all backup generations, divided by the total time to deduplicate that data. The cumulative average throughput is used to provide a summarised view of performance over time.

Next, we examine the server-side or cross-user deduplication throughput. We note that while client-side deduplication in Stage 2 is metadata-only based, the server-side deduplication in Stage 4 performs not only cross-user deduplication but also, in a real implementation, would store the new data to each client's data storage containers. As such, overall throughput would be limited to the data transfer rate on the system to long-term storage media like hard disk drives, which today range around 300 MiB/second. For our trace datasets, it is important that the metadata operations re-

quired for server-side deduplication do not slow down Stage 4; that is, they are at least 300 MiB/second. But this also implies that throughput at scale will be limited to the system HDD data transfer rate.

6.6.1 FSL Single-processor Throughput

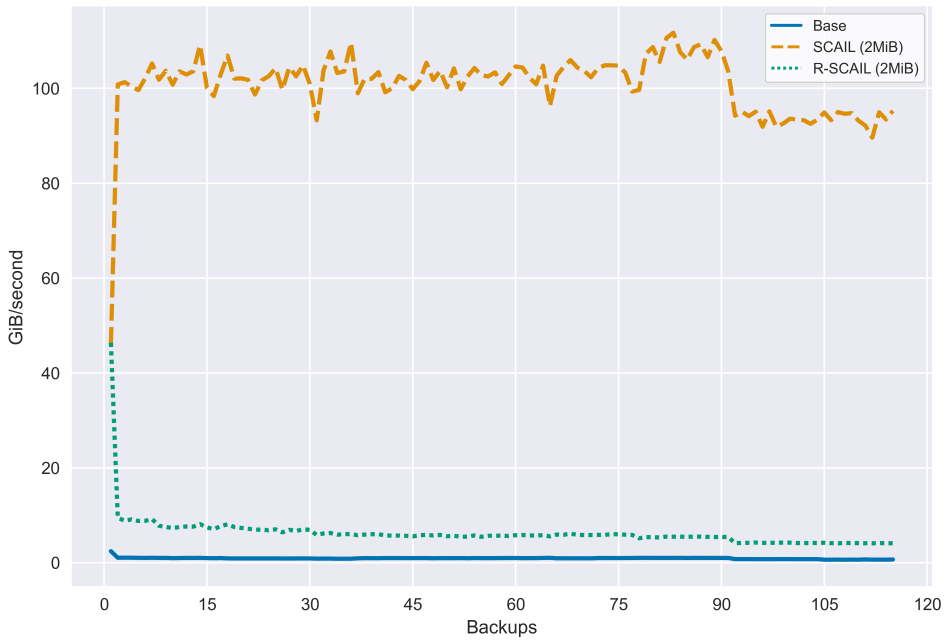


Figure 6.15: FSL Dataset: Single-Processor Throughput for Client-Side Deduplication. SCAIL and R-SCAIL significantly outperform Base. While R-SCAIL is much slower than SCAIL, it is faster than Base.

Client-side Deduplication Throughput

Figure 6.15 and the description below encapsulate the throughput dynamics, highlighting the computational demands and efficiencies inherent in each scheme on the FSL Dataset.

Base: Exhibiting the lowest throughput among the evaluated schemes at a cumulative average of 0.92 GiB/second, Base incurs a significant computational overhead for client-side deduplication due to the requirement of performing 6.9 bil-

lion lookups into the chunk index. This index grows from 28 million to 50 million elements over the course of the 115 backups. The throughput in the first backup is slightly higher than the ongoing throughput since the index is initially empty. While offering client-side deduplication at the chunk level, this granular approach is computationally intensive, making it the most demanding of processing resources among the three evaluated schemes.

SCAIL: SCAIL registers the highest throughput of the evaluated schemes. It leverages a higher-order deduplication process using metachunk fingerprints, which are much less numerous (91,000), and many fewer lookups (15 million) for the 115 backups, and consequently is much quicker to process. This highly leveraged memory-based index system facilitates a significantly less resource-intensive lookup process than Base's chunk-based index. SCAIL's throughput of 100.91 GiB/second is more than 100 times faster in cumulative average throughput than Base.

R-SCAIL: This scheme also uses the MFP Index like SCAIL, eliminating any previously saved segments from further consideration. Next, it adds an additional step of segment resemblance, giving chunk-level deduplication for the remaining segments. While this effectively reduces RSD by 97% (see Subsection 6.5.1), it requires disk-based lookups and the generation of larger "Missing" recipes since they include CFP's as well as MFPs. This additional step causes R-SCAIL to slow down after the first backup due to the disk I/O involved. Even though it performs 2,850 reads of metadata container manifests, it performs faster than the exclusively memory-based Base algorithm because it efficiently loads these recipes for deduplication and is able to avoid most of the chunk-level comparisons that the Base must perform. Although it is slower than SCAIL at 5.85 GiB/second cumulative average throughput, R-SCAIL markedly surpasses Base throughput by a factor of more than 6.

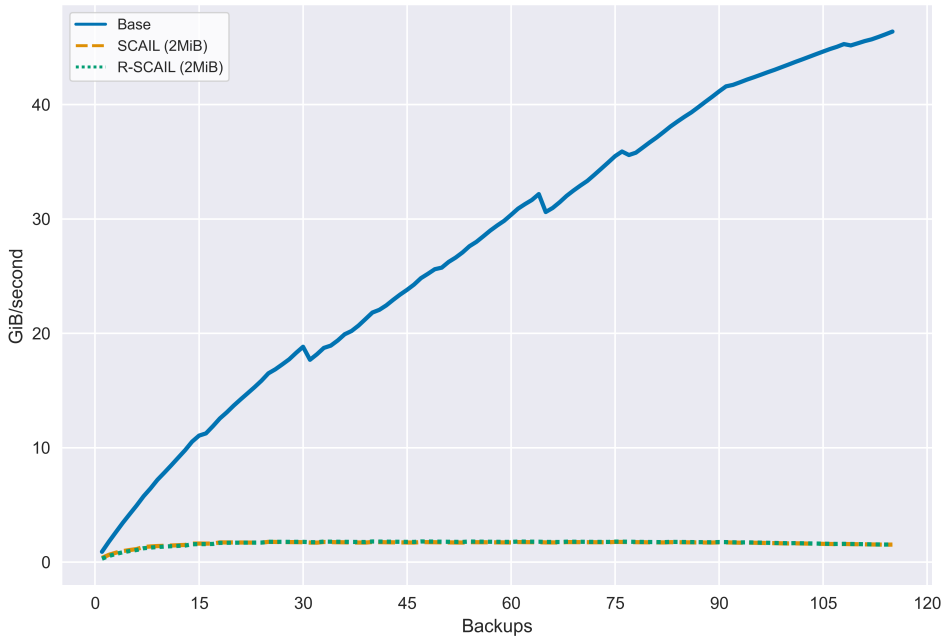


Figure 6.16: FSL Dataset: Single-Processor Cumulative Average Throughput for Server-Side Deduplication. Base substantially outperforms SCAIL and R-SCAIL.

Server-side Deduplication Throughput

As shown in Figure 6.16, Base at 46.3 GiB/second substantially outperforms SCAIL and R-SCAIL's 1.5 GiB/second. We note again that this throughput is for metadata operations only, and at this scale of data, Base's CFP Index can be memory-based rather than disk-based, which would be orders of magnitude slower. Meanwhile, SCAIL and R-SCAIL's SCI Index and Metadata containers are disk-based but are above the transfer rate to HDD, so they should not slow down Stage 4.

6.6.2 MS Single-processor Throughput

Client-side Deduplication Throughput

This subsection explores the throughput performance of client-side deduplication for the MS dataset, showing the per-backup throughput in Figure 6.17. We also calculate the cumulative average throughput for each scheme's efficiency over time.

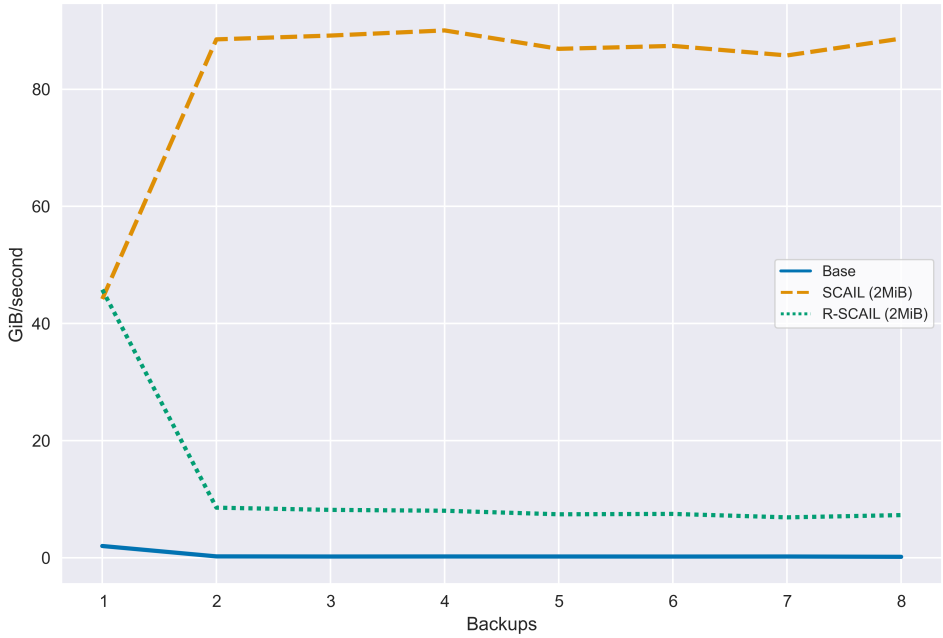


Figure 6.17: MS Dataset: Throughput for Client-Side Deduplication. All schemes use 8 KiB chunks, SCAIL and R-SCAIL use 2 MiB segments. The chart depicts the throughput of Base, SCAIL, and R-SCAIL, highlighting SCAIL's substantial lead and R-SCAIL's notable performance over Base.

Base: The Base scheme's cumulative average throughput is the lowest, at 0.2 GiB/s, which reflects its intensive computational overhead across all backups. The first backup has a slightly higher throughput since the index is empty at this point. The subsequent low throughput reflects the high number of lookups (7.3 billion) into the large (370 million entry) index, which must be performed for exact, chunk-level deduplication.

SCAIL: SCAIL achieves a cumulative average throughput of 75.5 GiB/s, 300 times faster than Base. This throughput indicates the high efficiency of SCAIL's deduplication process and its effective utilisation (12.7 million lookups) of a small metachunk fingerprint index (2.2 million entries) that significantly reduces client-side deduplication processing time.

R-SCAIL: With a cumulative average throughput of 8.9 GiB/s, PR-SCAIL performs 36 times faster than the Base scheme, though it does not match the throughput level

of SCAIL. Even though it is disk-based and performs 25,000 metadata container manifest reads, it maintains a substantial improvement over Base, validating the efficiency of its resemblance-based deduplication method.

The cumulative average throughputs underscore the superior scalability of SCAIL for the MS dataset. R-SCAIL also demonstrates enhanced throughput compared to Base, albeit to a lesser extent than SCAIL. These metrics reflect the raw throughput capabilities of each scheme and their relative efficiencies in a client-side deduplication context.

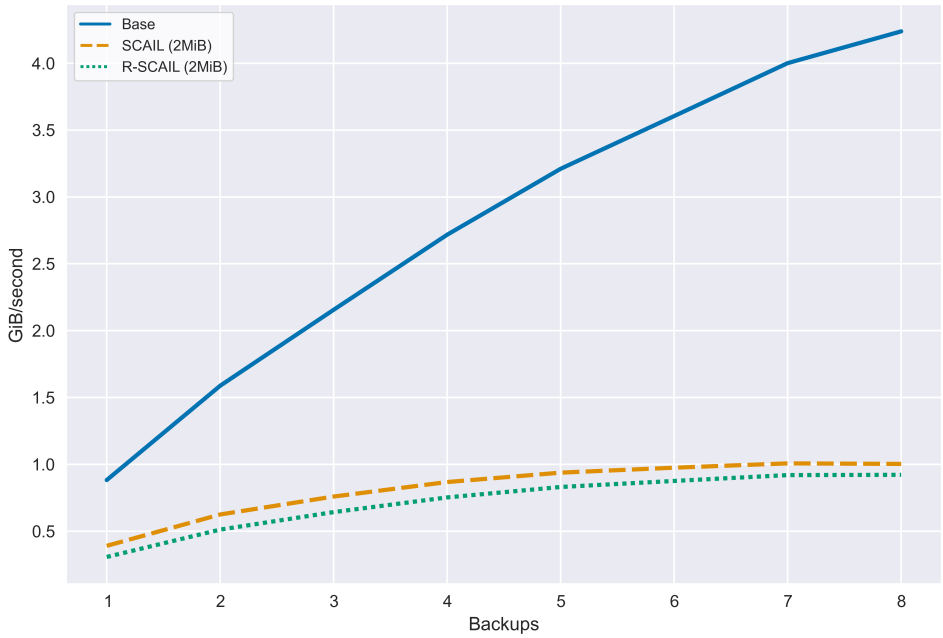


Figure 6.18: MS Dataset: Throughput for Server-Side Deduplication. Base outperforms SCAIL, and R-SCAIL, but all schemes show throughput above HDD transfer rates.

Server-side Deduplication Throughput

In Figure 6.18, we see that Base is over four times faster than SCAIL and R-SCAIL's 1 GiB/second of server-side deduplication throughput with these trace-based results, but note that server-side deduplication throughput of would be limited to data transfer rates in a real (non-trace) implementation.

6.6.3 Summary of Single Processor Throughput Analysis

The evaluation of single processor throughput for the Base, SCAIL, and R-SCAIL deduplication schemes has underscored the significance of throughput as a pivotal performance metric. Our analysis revealed that SCAIL and R-SCAIL schemes significantly outperform the Base scheme in terms of client-side deduplication throughput across both FSL and MS datasets.

The SCAIL scheme demonstrated remarkable efficiency, leveraging metachunk fingerprints for a less resource-intensive lookup process. This approach facilitated a throughput that was, 100 to 300 times faster than the Base scheme, highlighting the potential for substantial performance improvements in practical deduplication scenarios. Alternatively, the R-SCAIL scheme, while not matching the throughput of SCAIL, still offered a significant improvement over the Base scheme by incorporating resemblance-based deduplication to reduce redundant segment data (RSD) effectively and ranged from 6 to 36 times faster.

Server-side throughput analysis further illustrated the limitations imposed by data transfer rates to long-term storage media. Despite these constraints, the throughput achieved by all schemes was found to be above the minimum transfer rates, suggesting that metadata operations do not pose a bottleneck in the server-side deduplication process.

Recognising the wide availability of multiprocessor servers, we next examine increases in throughput in these systems.

6.7 Multiprocessor Throughput

In this analysis, we focus on P-SCAIL and PR-SCAIL schemes, which in the previous single-processor throughput section, have been shown to significantly outperform the Base scheme. With a 16-processor setup, we measure the effect of segment size on throughput. The SCAIL multiprocessor approach utilises task-based parallelism to

enhance client-side deduplication, while task and data parallelism are used in server-side deduplication.

The accompanying charts present instantaneous throughput for each backup instance to assess performance over the course of backing up the dataset. However, we employ the cumulative average throughput when discussing the schemes.

6.7.1 FSL Multiprocessor Throughput

For multiprocessor throughput on the FSL dataset, our analysis pivots to the performance of P-SCAIL and PR-SCAIL under different segment sizes, given their comprehensive performance advantage over the Base scheme. Utilizing a 16-processor setup simulated with the Ray framework (see Section 4.7), we aim to examine the throughput behaviour across varying segment sizes and the implications of task-based parallelism on client-side deduplication.

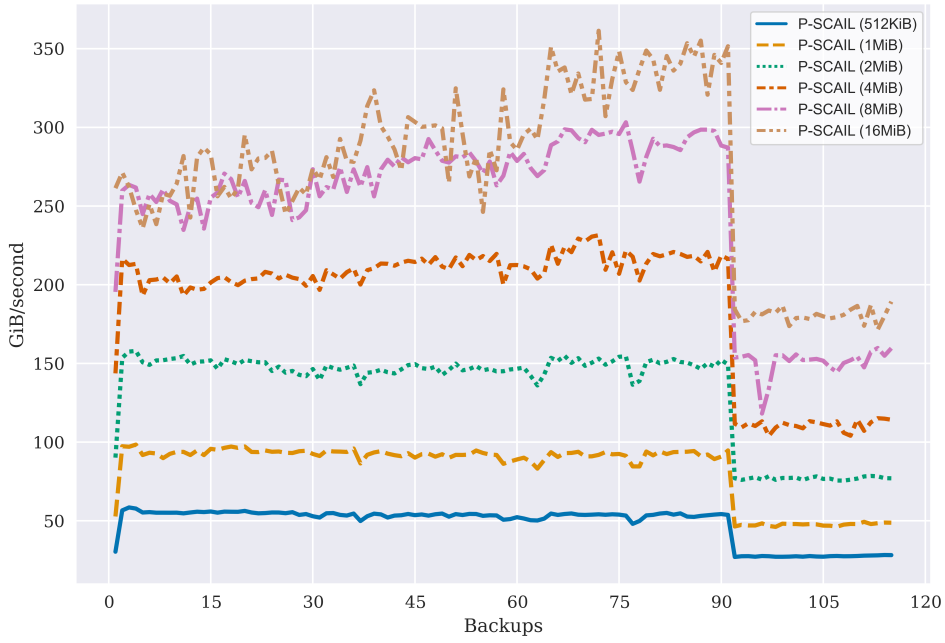


Figure 6.19: FSL Dataset: Multiprocessor Throughput Performance of P-SCAIL with Different Segment Sizes. Larger segment sizes result in higher throughput. A dropoff in throughput after 95 backups is caused by reduced logical data submitted for backup, which can be observed in Figure 6.20.

Segment Size Effect On Throughput

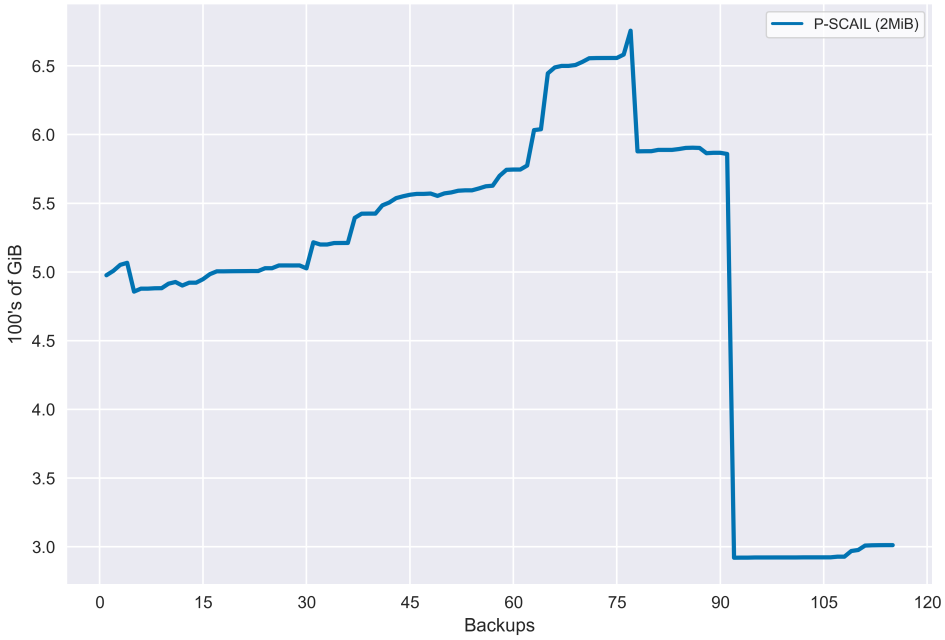


Figure 6.20: FSL Dataset: Logical Size of Backup Volume for each Backup Generation. After the 95th backup, the volume of backup data falls off. This corresponds to the reduced throughput observed in Figure 6.19.

Table 6.1: FSL Dataset: Client-side Deduplication Throughput with 16 Processors by Segment Size for P-SCAIL and PR-SCAIL

Segment Size	512 KiB	1 MiB	2 MiB	4 MiB	8 MiB	16 MiB
P-SCAIL Throughput (GiB/s)	47.9	82.3	132.4	189.3	247.9	273.5
PR-SCAIL Throughput (GiB/s)	15.2	21.5	19.8	27.1	18.8	16.5

We observed that the impact of segment size on throughput for PR-SCAIL was limited; thus, our focus is on P-SCAIL where differences are more pronounced. The chart in Figure 6.19 shows increasing segment size increases throughput for P-SCAIL. Table 6.1 summarises the cumulative average throughput for each SCAIL scheme, where the cumulative average throughput is the total volume of data for all backups, divided by the time taken to deduplicate the data. We also observed that the logical volume of data presented to the server for backup, as shown in Figure 6.20, influenced throughput. As the logical volume was reduced, throughput also fell. This reflects that the pro-

cessing time remains relatively stable, so if logical volume falls, throughput will also fall. Conversely, increased logical volume tends to increase throughput in P-SCAIL.

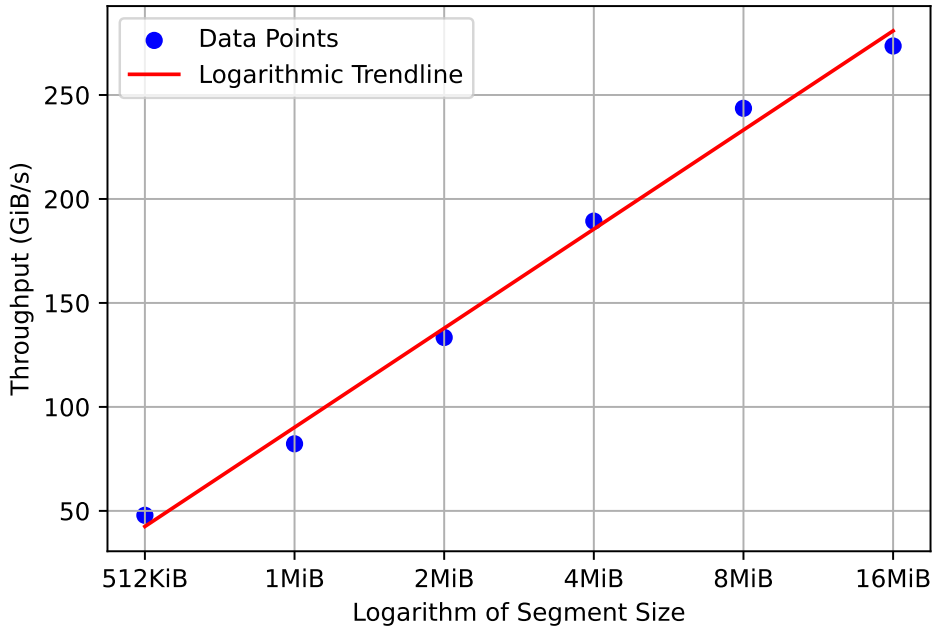


Figure 6.21: FSL Dataset: Throughput in GiB/second as a function of segment size, showing observed (dots) and predicted (line) percentage of RSD across a range of segment sizes.

Quantifying Segment Size Effect

In Figure 6.21, we chart client-side deduplication throughput for varying segment sizes. We found a curve-fitting logarithmic formula that maps segment size to the expected throughput to be:

$$\text{throughput} = 68.771 \times \log(\text{segment size}) + 90.170. \tag{6.1}$$

Alternatively, P-SCAIL has almost no RSD, but has lower throughput than P-SCAIL. We find that segment size does not have much impact on throughput with PR-SCAIL as well.

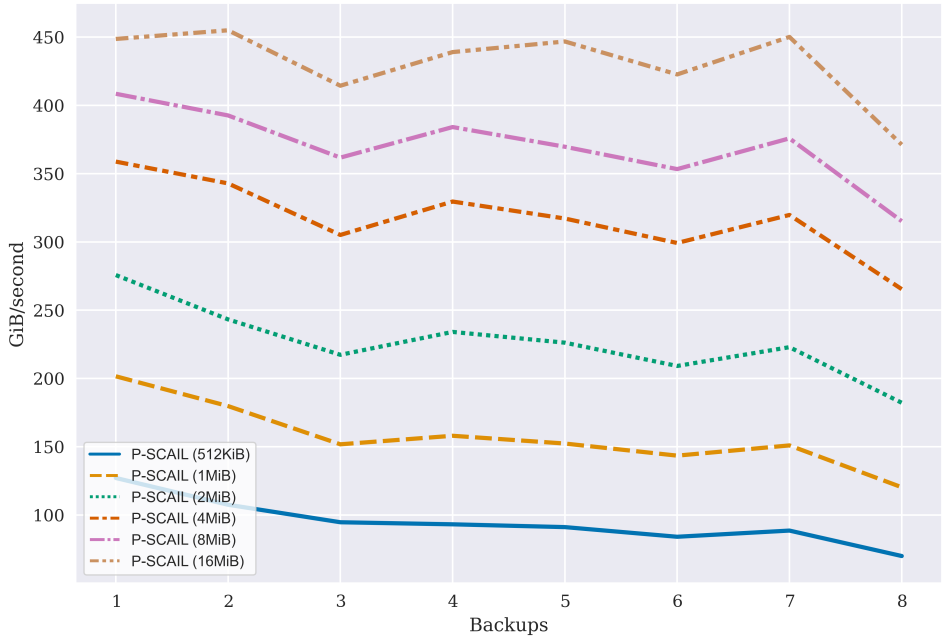


Figure 6.22: MS Dataset: Multiprocessor Throughput Performance of P-SCAIL with Different Segment Sizes. Larger segments produce higher throughput. Throughput also falls gradually for all segment sizes as the volume of data to be backed up falls, as shown in Figure 6.23.

6.7.2 MS Multiprocessor Throughput

The multiprocessor throughput for the MS dataset mirrors the methodology applied to the FSL dataset, emphasizing the performance under a 16-processor configuration. This analysis excludes the Base scheme, focusing on the throughput of P-SCAIL with varying segment sizes.

Segment Size Effect On Throughput

The instantaneous throughput for each backup is graphically represented in Figure 6.22, but our discussion below references the cumulative average throughput (total volume of client data in all backup generations divided by the time taken to deduplicate it), as presented in Table 6.2. This metric offers a more consistent performance indicator over the entire backup sequence.

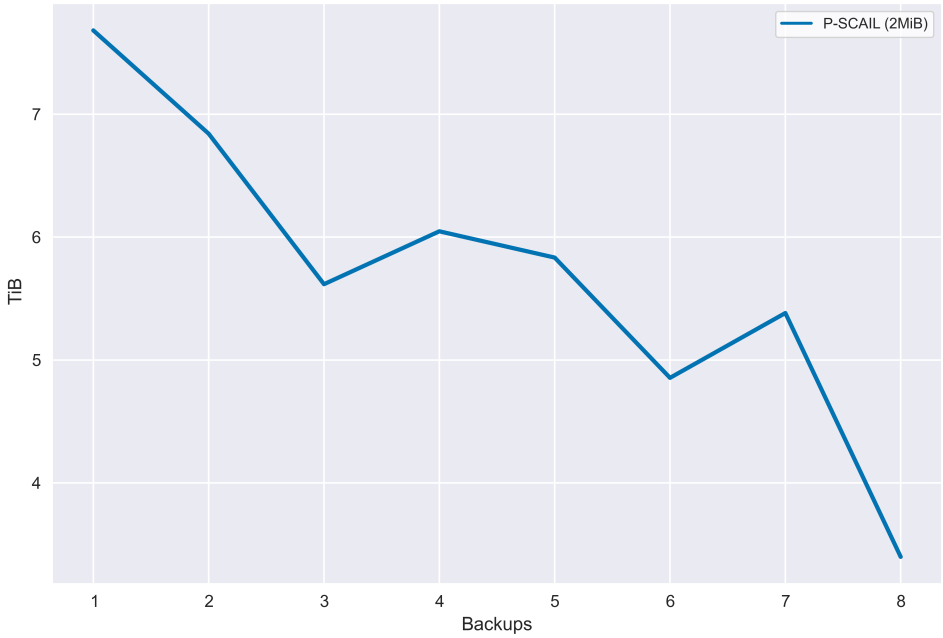


Figure 6.23: MS Dataset: Logical Size of Backup Volume for each Backup Generation. The chart shows that the logical data presented to the server for backup falls from 7.7 TiB on the first backup to 3.4 TiB on the last.

Table 6.2: MS Dataset: Client-side Deduplication Throughput with 16 Processors by Segment Size for P-SCAIL and PR-SCAIL

Segment Size	512 KiB	1 MiB	2 MiB	4 MiB	8 MiB	16 MiB
P-SCAIL Throughput (GiB/s)	95.2	158.9	228.9	320.7	373.9	434.2
PR-SCAIL Throughput (GiB/s)	37.3	58.1	79.3	91.8	91.8	79.0

Quantify Segment Size Effect

In Figure 6.24, we chart these results. We found a curve-fitting logarithmic formula that mapped segment size to the expected throughput for client-side deduplication throughput data to:

$$\text{throughput} = 105.903 \times \log(\text{segment size}) + 169.513. \tag{6.2}$$

We found that segment size does not substantially impact client-side deduplication throughput for PR-SCAIL.

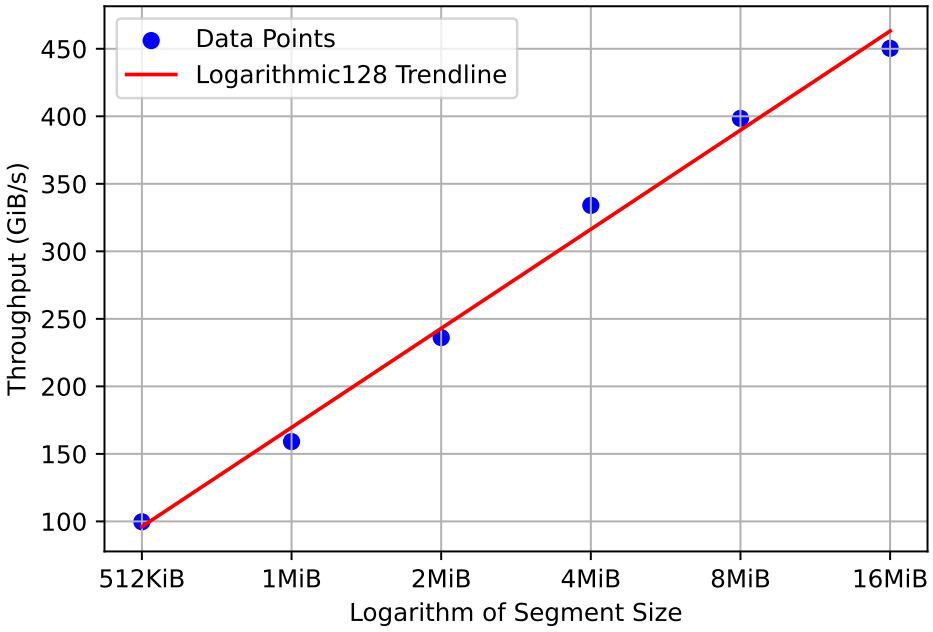


Figure 6.24: FSL Dataset: Client-side deduplication throughput in GiB/second as a function of segment size, showing observed (dots) and predicted (line) percentage of RSD across a range of segment sizes.

6.7.3 Summary of Multiprocessor Analysis

Our analysis shows that for both FSL and MS datasets, increasing segment sizes in a 16-processor setup boosts P-SCAIL throughput. The largest segments for P-SCAIL, reached the highest throughput but with more data to upload, as described in Section 6.5. Smaller segment sizes showed lower throughput and lower upload volumes.

We note that client-side deduplication throughput for the FSL dataset is about half that of the MS dataset. This is because there are only 8 clients in the FSL dataset, so, during client-side deduplication, half of the processors are idle. Also, we find that the throughput trends follow the volume of logical data; a decrease in data volume over time leads to reduced throughput. This points to the system's suitability for high-volume backup workloads.

With these throughput findings presented, we now turn to cost analysis.

6.8 Comparative Analysis of Server Component Costs

In the preceding analysis, we presented the memory and server data and metadata storage requirements for the Base, P-SCAIL and PR-SCAIL schemes. In this section we integrate and summarise this data. This analysis will not only reveal the cost efficiency of the SCAIL family compared to Base but also give insight into the practicality of deploying these systems.

We use the costs outlined in Table 4.2, which we summarise here as $2.375 per GiB memory, $0.0617 per GiB of SSD and $0.0154 per GiB of HDD. For the chart values for Data, we multiply the volume of unique chunk data stored on the server by the HDD cost. For Metadata, we multiply the volume for the components outlined as Metadata in Section 6.4 (Lookup Index, SCI Bins, Recipes, Metachunks) by the SSD cost. For Memory, we use the total volume presented in Section 6.3 and multiply by memory cost. All results are based on 8 KiB chunks and 2 MiB segments for P-SCAIL and PR-SCAIL. For more specific costing details, see Subsection 4.7.3.

6.8.1 FSL Component Costs

For the FSL Dataset, the Base scheme presents a traditional approach with associated costs across data storage, index management, and cumulative metadata handling. The total cost for the Base setup over the 115 backups amounts to $32.83, with the largest share attributed to the metadata cost at $22.84, reflecting the SSD storage costs for the large volume of File and Key Recipes and the disk storage for the CFP Index.

With P-SCAIL and PR-SCAIL, there is a marked reduction in both index and metadata costs, reflecting the efficiency of segment-level deduplication. The total cost is significantly lower at around $7.94, attributing a 75% decrease in cost compared to Base. This reduction underscores the optimised use of metachunk-only fingerprint lookup, which incurs substantially less cost at $0.68 for index management and a minimised metadata overhead at $0.60.

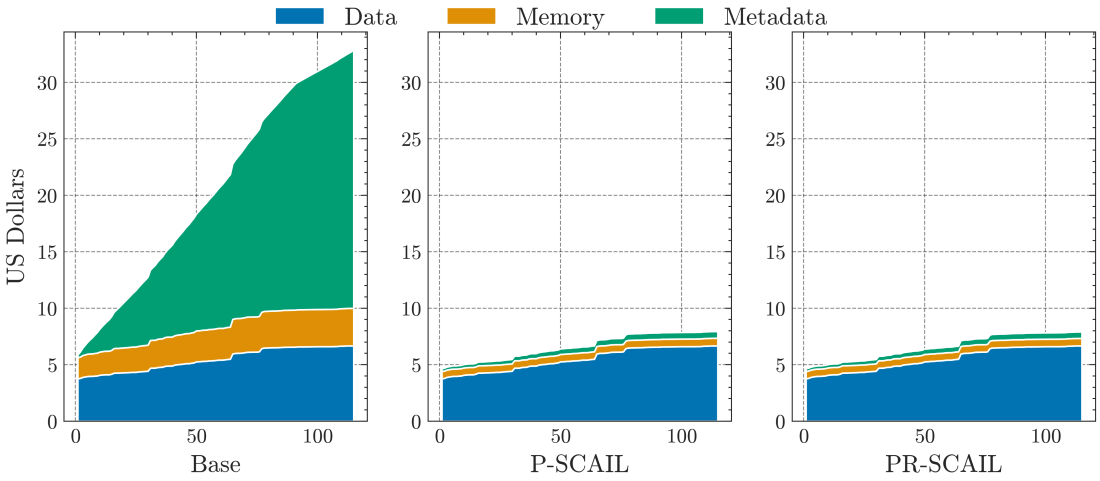


Figure 6.25: FSL Dataset: Cumulative backup costs broken down by data, metadata and memory components, for the Base, P-SCAIL and PR-SCAIL schemes.

Figure 6.25 visually encapsulates these cost differentials, showcasing the stark contrast between the Base scheme's escalating costs and the flattened cost curve observed in both P-SCAIL and PR-SCAIL. This chart solidifies the comparative analysis, highlighting the economic advantages of adopting more advanced deduplication strategies.

6.8.2 MS Component Costs

With the MS dataset, we explore the cost implications of a high-demand corporate environment, reflecting the routine and comprehensive data management needs of a large-scale operation. This dataset presents a formidable challenge, with its substantial volume and frequent updates necessitating a robust and efficient deduplication strategy. The total cost analysis for the Base scheme reveals a significant expenditure across data storage, index management, and metadata storage, culminating in a total cost of $94.09. The substantial index cost of $24.60, alongside the metadata cost of $26.99, illustrates the considerable resources required to manage the backup process over eight weeks for a large number of workstations.

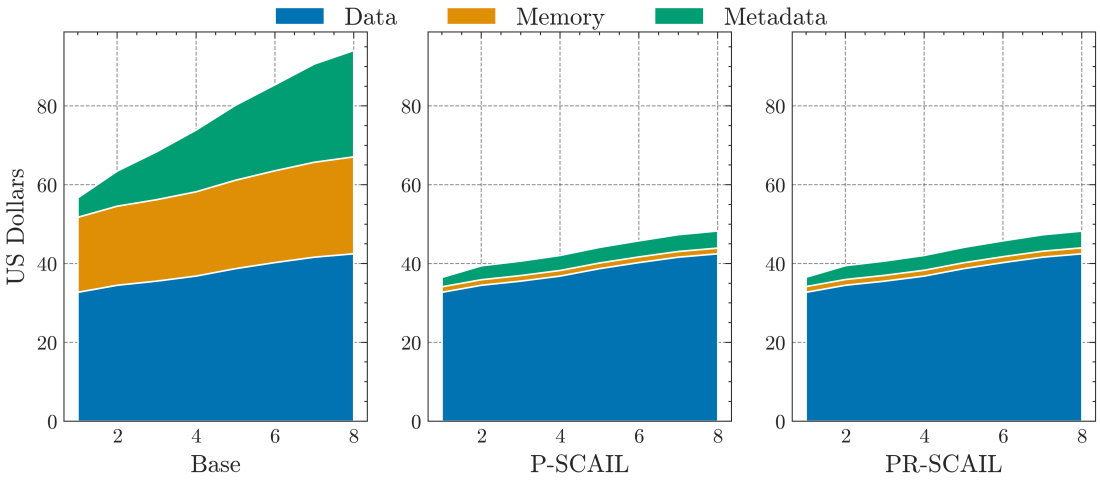


Figure 6.26: MS Dataset: Stacked area chart showing component costs. All schemes use 8 KiB chunks, and P-SCAIL and PR-SCAIL use 2 MiB segments.

Both PR-SCAIL and PR-SCAIL demonstrate a remarkable reduction in index and metadata costs. The total costs for each SCAIL scheme are around $48.29, and index and metadata expenses are around $1.45 and $4.34, respectively. This translates to a total cost reduction of nearly 49% compared to the Base scheme.

Figure 6.26 provides a visual representation of these costs, underscoring the economic advantage of adopting advanced deduplication strategies in environments characterised by large data volumes and a high degree of cross-user data redundancy.

6.8.3 Summary of Cost Findings

Our cost analysis for the FSL and MS datasets highlights the economic impact of deduplication schemes across two distinct backup environments. The Base scheme, while serving as a benchmark, incurs higher costs across memory and metadata. Through innovative deduplication strategies, the P-SCAIL and PR-SCAIL schemes significantly lower these costs.

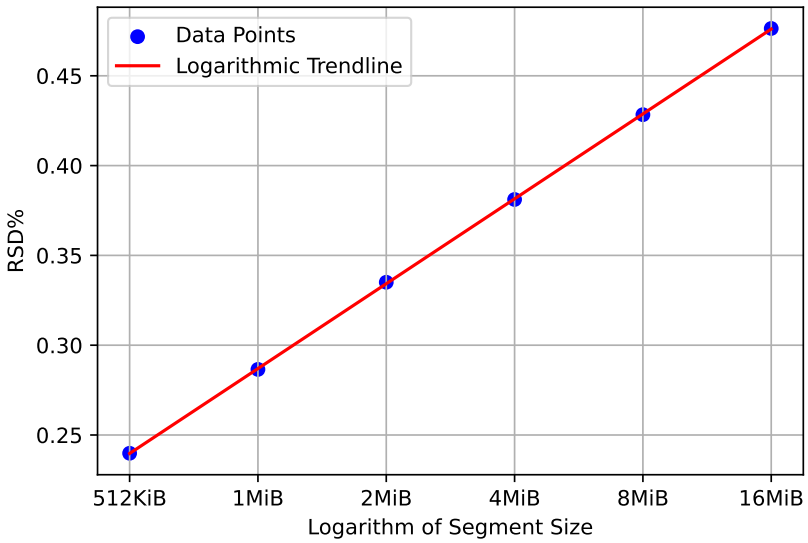
The cost reductions seen in the FSL dataset are consistent in the MS dataset, albeit with different magnitudes due to the distinct nature of the data and the frequency of

backups. The P-SCAIL and PR-SCAIL schemes not only provide a technical upgrade over the Base scheme but also present a compelling case for cost savings in long-term data management strategies. These findings suggest that the SCAIL family of algorithms offers a scalable, cost-effective solution for enterprise-level backup systems.

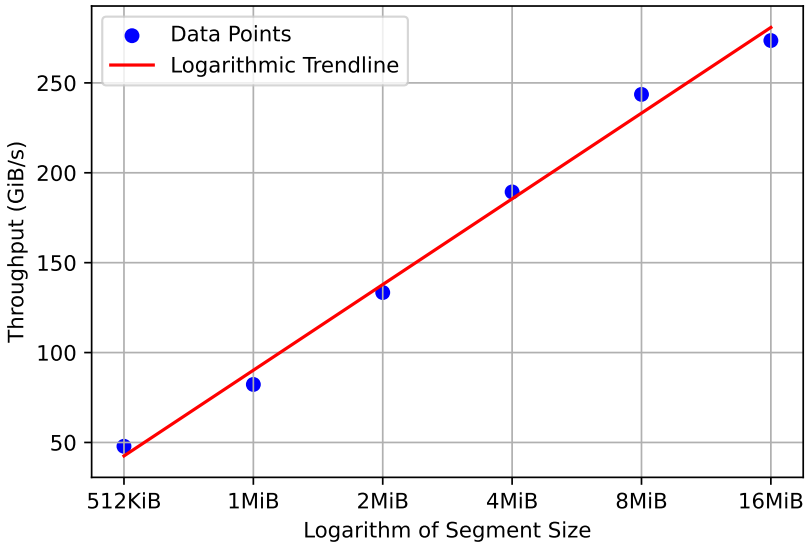
As we conclude this costing evaluation, it can be seen that the choice of deduplication scheme can significantly influence the total cost of ownership for data backup systems. The insights garnered here will serve as a foundation for our overall comparative findings, which we present next.

6.9 Comparative Findings

In this section, we synthesise insights from the extensive comparison of Base and the SCAIL family of algorithms, focusing on their efficiency, performance and suitability for different workloads. Across various metrics ranging from memory and storage requirements to upload volume and throughput – the aim has been to explore the strengths and limitations inherent in each approach. This comparison not only highlights the trade-offs between resource utilisation and system performance but also sets the stage for identifying preferred deployment scenarios based on specific requirements and constraints.



(a) Percentage of RSD



(b) Client-side Deduplication Throughput

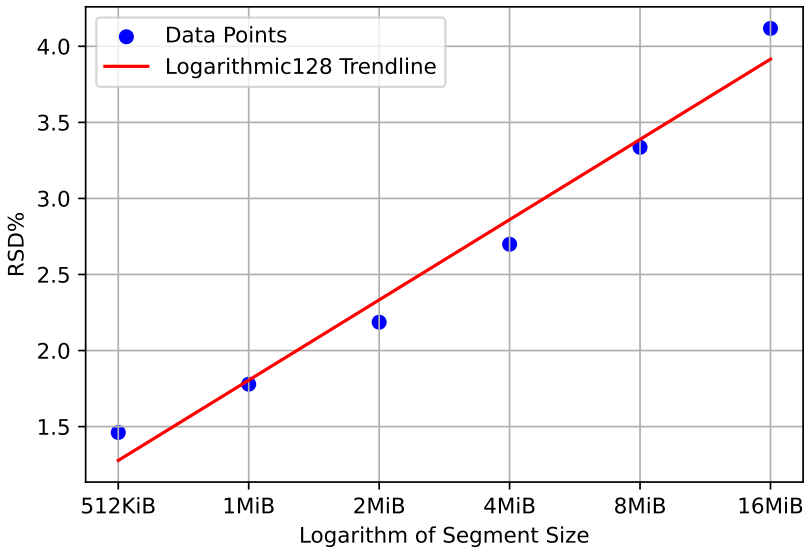
Figure 6.27: FSL Dataset: These two charts allow you to compare, for a given segment size, the expected RSD% (top chart) and Throughput (bottom chart).

6.9.1 Efficiency and Performance Trade-offs

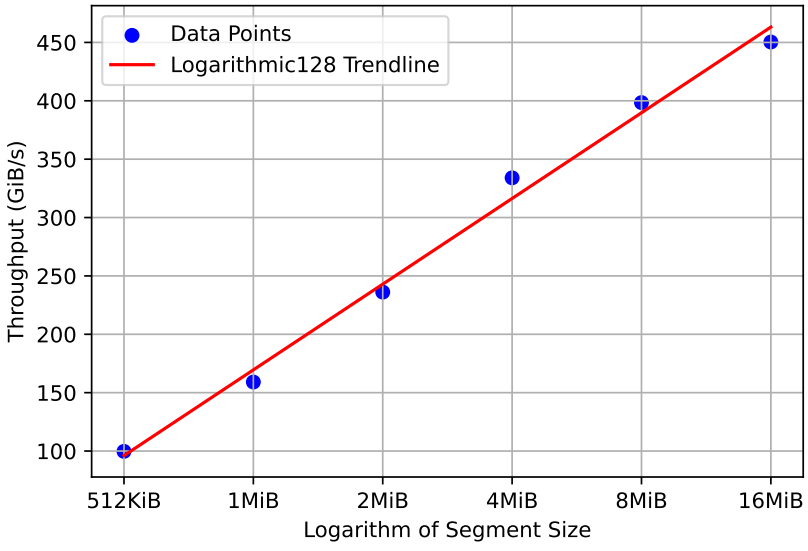
In deduplication, striking a balance between resource efficiency and system performance is pivotal. Our comparative study has examined some trade-offs that come into play with different deduplication schemes.

Memory and Storage Efficiency: The SCAIL family of algorithms exhibit a pronounced advantage in memory and storage efficiency over the Base scheme. This is especially critical when managing large-scale workloads where resource optimisation is directly linked to system scalability and cost-effectiveness. The memory footprint of the SCAIL schemes, being substantially smaller, offers a sustainable model for expanding data management operations while reducing associated costs.

Server-side Deduplication Throughput: Server-side deduplication throughput for the SCAIL family remains competent, adequately surpassing the throughput constraints of disk-bound data transfers to HDD storage. This efficiency is paramount in maintaining acceptable backup windows, making SCAIL schemes suitable for enterprise-level deployments where time and data integrity are of the essence.



(a) Percentage of RSD



(b) Client-side Deduplication Throughput

Figure 6.28: MS Dataset: These two charts allow you to compare, for a given segment size, the expected RSD% (top chart) and Throughput (bottom chart).

Client-side Deduplication Throughput and Upload Volume: For client-side deduplication, P-SCAIL demonstrates a remarkable throughput that far exceeds the Base scheme, aligning well with high-performance requirements. However, this comes at the cost of increased RSD volume. While this may represent a small fraction of total uploads, the cumulative effect over time can impact short term storage demands and network traffic of data backup solutions.

We collect the findings from the Upload Volume and Multiprocessor Throughput sections above to present the RSD vs Throughput trade-offs for P-SCAIL for each dataset. Given a percentage of RSD per backup that can be tolerated, the top charts in Figure 6.27 and Figure 6.28 (which are detailed in Subsection 6.5.2) can be used to find the resulting segment size. With this segment size identified, the expected throughput can be read from the bottom charts (detailed in Subsection 6.7.2).

Our analysis shows a clear trade-off between the percentage of RSD and achievable throughput in the P-SCAIL scheme. System administrators can use the provided charts to determine an appropriate segment size that balances these factors according to their specific RSD tolerance. The choice of segment size is a strategic decision that influences immediate throughput and long-term aggregate upload volume from RSD.

6.9.2 Suitability for Different Workloads

FSL vs MS Performance With 16 processor configurations, we note that client-side deduplication throughput for the FSL data is about 1/2 that of the MS dataset. This reflects that only eight clients are in the FSL dataset, compared to 128 for the MS dataset. So, if the workload has fewer clients than the number of processors dedicated to backup, there will be system inefficiencies. SCAIL family schemes always require less server storage than the Base scheme, and this is especially prominent in longer-term backups, as un-deduplicated File and Key Recipes accumulate a significant volume of space on the server.

Dataset Size We also found that the larger the dataset, the larger the advantage SCAIL scheme over the Base scheme.

6.9.3 Final Recommendations

We compare the SCAIL family algorithm by three types of upload volume constraints:

- Baseline: Maintain the same approximate upload volume as Base.
- Unlimited: Prioritise throughput over upload volume.
- Minimise: Upload the smallest volume possible.

For example, examining the first row in Table 6.3, if the primary requirement is to maintain the upload volume for a traditional encrypted deduplication system like Base, then from Figure 6.9, we can see that a 4 MiB segment size will upload effectively the same volume. From the comparison Figure 6.27 above, we can see that it will produce an estimated client-side deduplication throughput of 189.3 GiB/second, and the upload volume will have 0.38% RSD.

Table 6.3: Recommended Schemes and Segment Sizes for Different Upload Constraint Scenarios.

Upload Constraint	Dataset	Scheme	Seg. Size	% Baseline	GiB/s	RSD percentage
Baseline	FSL	P-SCAIL	4 MiB	-2.1%	189.3	0.38%
	MS	P-SCAIL	512 KiB	1.10%	95.2	1.50%
Unlimited	FSL	P-SCAIL	16 MiB	21.5%	273.6	0.47%
	MS	P-SCAIL	16 MiB	134.0%	434.0	4.10%
Minimise	FSL	PR-SCAIL	4 MiB	-43.6%	27.1	0.02%
	MS	PR-SCAIL	8 MiB	-7%	91.8	0.50%

Chapter 7

Conclusion

Encrypted deduplication backup systems are widely used, but face significant challenges. These include excessive metadata accumulation for long-term backups, limits to the volume of data that can be accepted in the system imposed by memory constraints, slow throughput caused by excessive disk I/O, and resource contention between competing backup jobs. Although various solutions have been proposed, existing approaches often struggle to balance efficient duplicate elimination with scalability, low memory overhead, and strong data privacy.

Recognising this gap in the field, we systematically investigated and developed novel approaches to encrypted deduplication to address these issues. First, we introduced **S**egment **C**hunks **A**nd **I**ndex **L**ocality (SCAIL), a novel hybrid two-phase deduplication system. To mitigate side-channel attacks, SCAIL performs client-side deduplication on metachunks privately for each client, avoiding cross-client deduplication at this phase. Using a metachunk-only deduplication index dramatically reduces the amount of memory required for client-side deduplication, enabling the deduplication index to be exclusively memory-based, even at large scales. This results in very fast throughput, significantly improving performance over existing methods. After uploading the chunks of missing metachunks and sorted lists of their fingerprints, SCAIL processes a multi-client batch backup using index locality to find and discard duplicate chunks before final storage to the server. This technique uses low amounts of memory and requires few disk I/Os at large scales.

Next, to enable backup servers to leverage the enhanced capacity introduced by SCAIL, we developed **P**arallel SCAIL (P-SCAIL), a parallel implementation that employs data and task parallelism. This advancement ensures that backup servers have the throughput needed to handle the increased data volumes.

Finally, we found that SCAIL and P-SCAIL produced redundant chunk uploads in certain scenarios. To overcome this limitation, we designed **P**arallel **R**esemblance SCAIL (PR-SCAIL), an extension of P-SCAIL that dramatically reduces excess uploads through segment resemblance-based chunk deduplication techniques, albeit at the cost of a reduction in throughput.

We rigorously evaluated each of these systems using real-world, trace backup datasets, demonstrating significant improvements in storage capacity, efficiency, throughput, and upload volume reduction.

7.1 Revisiting Our Objectives

In this thesis, our objectives aim to enhance the performance and capacity of client-server deduplication systems. The objectives guided the development and evaluation of our deduplication approaches, each designed to address specific challenges in large-scale, secure data backup environments. The main objectives of this research are:

1. **Reduce Server Storage**: Reduce server storage requirements by deduplicating metadata as well as performing exact, chunk-level deduplication at scale.
2. **Reduce Memory Requirements**: Reduce server memory requirements associated with processing client-side and server-side deduplication, aiming to mitigate disk bottlenecks and support petabyte scalability on a single server.
3. **Fast Client-side Deduplication**: Enable high-speed throughput for client-side deduplication, demonstrating the feasibility of processing client backups in the petabyte-scale of unique (deduplicated) data.
4. **Ensure Data Privacy**: Provide strong data privacy guarantees against brute force

and other attacks, and ensure resistance to side-channel attacks.

5. **Reduce Resource Contention**: Reduce, and if possible, eliminate resource contention during client-side and server-side deduplication.
6. **Increase Throughput with Parallelism**: Increase deduplication throughput by leveraging parallelism on multiprocessor systems.
7. **Reduce Upload Volume**: Achieve reduced upload volumes by incorporating metadata deduplication and resemblance techniques.

With the given objectives, we provide a detailed examination of how they were addressed in the design, implementation, and testing phases of our work. In the following, we discuss the specific techniques and approaches applied to meet each objective, highlighting the key results and insights.

Objective 1. Reduce Server Storage: SCAIL, generates *metachunks* from encrypted chunk metadata based on the Metadedup design. Deduplicating these metachunks achieved a significant reduction in server metadata storage—97% for the FSL dataset and 86% for the MS dataset (see evaluation results in Section 6.4). This approach also reduced overall server storage requirements compared to traditional chunk-based deduplication by 44% for the FSL dataset and 11% for the MS dataset.

However, the storage reductions above assume chunk-level deduplication is performed on the server. Unlike Metadedup, SCAIL's MFP-only index can only perform deduplication at the segment level, not the chunk level. Deduplicating only at the segment or metachunk level would miss the additional compression benefits of chunk-level deduplication. To maximise storage efficiency, we incorporated exact, cross-user, chunk-level deduplication for server-side deduplication. This approach captures redundancy at the finest granularity, ensuring that identical chunks within different segments are deduplicated.

Performing exact deduplication as the final step before storing data on the server frees us to employ client-side deduplication approaches that accelerate throughput.

These include coarse-grained (segment-level) deduplication in Chapters 3 and 4, and chunk-level, near-exact client-side deduplication in Chapter 5. Combining efficient server-side chunk-level deduplication with these client-side strategies achieves storage savings (Objective 1).

Objective 2. Reduce Memory Requirements: Having dramatically reduced memory requirements for client-side deduplication in Objective 3, we still needed to implement chunk-level cross-user deduplication for server-side deduplication. Memory requirements for traditional chunk-level encrypted deduplication grow with the size of the accumulated unique data identified by the system. To address this, we utilised the batch-oriented SCI design (see Subsection 4.7.1). This approach required only a single, modestly-sized cache whose size was independent of the number of clients, and independent of the volume of data in a batch. That is, adding more clients to a batch did not require any increase in memory requirements (see Subsection 3.3.4).

Objective 3. Fast Client-side Deduplication: The MFP-only index enables us to perform memory only segment-level client-side deduplication theoretically up to petabyte-scale datasets, as calculated in Section 3.4. This achieved exact (albeit at the segment-level), high-throughput deduplication, but did cause the occasional upload of some previously saved chunks.

Objective 4. Ensure Data Privacy: Ensuring the privacy of client data in a cloud backup system is of the highest priority. We, therefore, aimed to provide strong data privacy guarantees against brute-force and other attacks and to mitigate side-channel attacks. This objective was achieved by employing the following techniques across the different schemes developed in the thesis.

- **Chunk and Metachunk Encryption** We leveraged the Message-Locked Encryption (MLE) and DupLESS approaches to secure data and metadata chunks (see

Section 3.10).

- **Small File Handling** To maintain the computational complexity required to protect against brute-force attacks, we use a stream of bytes rather than individual files to create metachunks (see Subsection 3.10.2).

- **Side-Channel Attacks** To mitigate side-channel attacks, we avoided cross-client deduplication during the client-side deduplication phase, deferring it to the server-side phase. This approach prevents the server from revealing the upload status of specific data chunks to clients, though it reduces the upload efficiency of client-side deduplication (see Subsection 3.10.3).

Objective 5. Reduce Resource Contention: For client-side deduplication, resource contention was reduced by using read-only access to memory-based Metachunk Fingerprint (MFP) index. This allows multiple clients to safely access the index simultaneously. After all client data is uploaded, deduplicated, and stored on disk, the index is updated in a single pass (see Section 4.2).

PR-SCAIL was also able to reduce resource contention in client-side deduplication since each client has their own data and metadata containers. This meant that a given process could work on a client's deduplication simultaneously without having to coordinate access to common data containers (see the discussion of parallelism in Stage 2 of PR-SCAIL, Subsection 5.6.2).

For server-side deduplication, using the concept of Sorted Deduplication's Sorted Chunk Indexing (SCI) allowed all clients to be grouped into large batches. This enabled the simultaneous deduplication of a large number of client backup streams in a single pass through the chunk-level disk-based index (see Subsection 3.3.4). This eliminated resource contention on the disk-based chunk-level index, which would have been caused by allowing simultaneous access from multiple clients. This lack of resource contention enabled an increase in throughput by taking advantage of multipro-

cessor systems, as laid out in Section 4.3.

Objective 6. Increase Throughput with Parallelism: In Chapters 4 and 5, we implemented task-based and data-based parallelism to increase system throughput. For client-side deduplication, we implemented task-based parallelism.

For P-SCAIL, we dedicated a processor for each client query in turn, to perform lookups into the segment-level index and achieved a throughput of up to 273 GiB/second for the FSL dataset and up to 435 GiB/second for the MS dataset, (see Tables 6.1 (FSL) and 6.2 (MS) for P-SCAIL).

For PR-SCAIL, we also implemented task-based parallelism, where the manifests of similar segments were loaded from client-specific metachunk containers. We were able to achieve near-exact chunk-level deduplication throughput of 27 GiB/second for the FSL dataset, and 91 GiB/second for the MS dataset.

For server-side deduplication, we employed a combination of task-based and data-based parallelism to increase deduplication throughput. Duplicate lookup in SCI was split between processors using data parallelism, with each processor working on a disjoint subrange of the total range of CFPs. Once duplicates had been found, we allocated chunks to containers using task-based parallelism, allocating a processor to each client stream. We achieved parallel processing throughput for server-side deduplication of up to 10 GiB/second for the FSL dataset and 6.9 GiB/second for the MS dataset, (see Subsection 4.7.1).

Objective 7. Reduce Upload Volume: In Chapter 5, we addressed the issue inherent in SCAIL's design of the occasional re-upload of chunks already stored on the server by introducing PR-SCAIL. By adding near-exact, chunk-level, resemblance-based deduplication to our exact, segment-based deduplication, we reduced the excess upload volume by 97% for the FSL dataset and 80% for the MS dataset (see Subsection 5.10.3). The memory requirements of this design are still very low, but checking for previously

uploaded chunks in the on-disk metadata containers slowed down client-side deduplication compared to segment-only (P-SCAIL) client-side deduplication.

7.2 Implications and Significance

Through the development of SCAIL, P-SCAIL, and PR-SCAIL, we have advanced the field of encrypted deduplication by introducing innovative indexing structures and deduplication techniques that increase efficiency, scalability, and security. Our work demonstrates that it is possible to achieve high throughput and reduce resource conflict and consumption without compromising data privacy. These contributions lay the groundwork for future research in designing next-generation deduplication systems in data storage and backup in secure environments.

Practical Applications: Organisations can implement our methods to improve the efficiency and security of their backup systems, particularly those handling petabyte-scale amounts of data. Increasing single-server capacity to the petabyte scale, with throughput performance to support it, and lowering operational costs, making it feasible for small and medium-sized enterprises to adopt robust encrypted deduplication backup solutions.

Theoretical Contributions: Our introduction of the MFP-only index and the hybrid two-phase deduplication approach facilitates tailored indexing approaches for client-side and server-side operations. Additionally, by combining exact *and* resemblance-based deduplication techniques, we enhance the theoretical understanding of the encrypted deduplication process. These advancements challenge existing assumptions regarding the trade-offs between deduplication granularity and system performance, enabling more flexible and efficient systems.

Influence on Future Research: Our work opens up new research directions, such as adaptive deduplication strategies that adjust to workload characteristics for improved performance. It encourages further exploration into hybrid deduplication methods and efficient indexing structures that can be targeted to specific workload requirements.

Alignment with Industry Trends: As data privacy regulations become stricter and data volumes grow rapidly, our solutions align with the industry's need for secure, scalable, and efficient data management systems. By providing strong data privacy guarantees without sacrificing performance or capacity, our work supports organisations in meeting regulatory compliance and protecting sensitive information.

7.3 Limitations

Our designs and evaluations have several limitations. We list the salient ones here for transparency and to guide further research.

Scalability and Memory Constraints: A core limitation across the SCAIL family is the reliance on in-memory data structures, particularly the MFP index. While this design significantly reduces memory consumption and eliminates disk I/O compared to traditional chunk fingerprint indexes at large scale, it still poses challenges for datasets exceeding multiple petabytes of unique data. In addition, PR-SCAIL adds a second index, the Representative Fingerprint (RFP), which is usually much smaller than the MFP index. It might be possible to mitigate the memory limitation by using an SSD-based index, and/or increasing metachunk size, which decreases the size of the MFP index.

Performance Trade-offs: SCAIL demonstrates significant improvements in metadata storage reduction and client-side deduplication speed. However, if a dataset is small

enough to be deduplicated with a memory-only hash table (say under 100 TB of deduplicated data), its use of the disk-based SCI technique for chunk-level server-side deduplication would be slower than the speed of the memory-based index.

PR-SCAIL's resemblance-based approach effectively mitigates Redundant Segment Data (RSD) uploads, but at the cost of reduced client-side deduplication throughput compared to P-SCAIL.

Data Characteristics and Workload Dependencies: The demonstrated efficiency and effectiveness of the SCAIL family are influenced by the nature of the backup workload and data characteristics. SCAIL's performance might not compare favorably for low-change datasets, since the amount of chunk-level data that must be deduplicated is potentially small after the first backup is completed. In this case, other indexing techniques, such as a disk-based hash table, may outperform SCI, as it requires a full pass through the entire index, whereas the hash table may need only a few lookups.

SCAIL and P-SCAIL's client-side deduplication is restricted to the metachunk level, which means duplicate chunks within different metachunks are not identified before uploading, causing write amplification. Similarly, restore operations, since they are of metachunk granularity, also exhibit read amplification. Adding another round-trip to the server, and enabling upload and download at the level of chunks rather than metachunks could mitigate this issue, but the performance and security implications of this approach should be weighed.

The evaluations in this research primarily rely on the FSL and MS datasets. These datasets, while widely used, may not capture the full spectrum of real-world data variations. Further investigation into diverse backup workloads could shed light on whether there are further scenarios where SCAIL-based algorithms are not appropriate.

Security and Privacy Considerations: The thesis strongly emphasises data privacy through Message-Locked Encryption (MLE) and DupLESS. However, our focus on an honest-but-curious adversary might not fully address potential vulnerabilities in real-world deployments with more sophisticated attackers. Exploring defences against attacks targeting access pattern leakage or employing advanced cryptographic techniques could reveal vulnerabilities.

System Management and Flexibility: Certain design choices, such as P-SCAIL's processor count dependency for client-side data partitioning, might restrict system flexibility and complicate management in a dynamic environment.

The batch-oriented nature of server-side deduplication in SCAIL could pose challenges for real-time or continuous backup scenarios that require immediate restore data availability. A method to ease this situation might be to enable multiple passes to be made through the SCI index simultaneously.

7.4 Future Work: Adaptive Client-side Deduplication

In our comparative analysis, the juxtaposition of the P-SCAIL system's high client-sided deduplication throughput (albeit with the accumulation of Redundant Segment Data (RSD)) against the PR-SCAIL system's lower throughput but minimal RSD highlighted the potential for an adaptive deduplication strategy. Our investigations, as documented in our publication and prototype implementations, have traditionally focused on employing only one of the P-SCAIL or PR-SCAIL systems at a time, examining their respective strengths and limitations. This raises an intriguing question: could an adaptive strategy be devised capable of tailoring deduplication techniques to backup workload characteristics, or even to individual clients? Such a strategy could potentially enhance throughput in addition to reducing RSD on a global scale.

Stage 2 Lookup Query: For the adaptive strategy to be viable, clients would need to use a PR-SCAIL-style lookup query, including Representative Fingerprint (RFP)'s in their Stage 2 lookup queries following the PR-SCAIL model, involving the upload of Chunk Fingerprint (CFP) recipes for each backup segment, beyond just the Metachunk Fingerprint (MFP) recipes. This enables the server to perform near-exact chunk-level client-side deduplication.

Resemblance Data Collection: Implementing the adaptive strategy necessitates accumulating segment resemblance data via the RFP Index during backup, whether it is used in subsequent backups or not. The RFP Index's size is typically half that of the already compact MFP index, and can be efficiently updated concurrently with the MFP Index once the processing of "Missing" Recipes have been processed.

Threshold Determination: In the lookup phase, the system always conducts MFP lookups to identify and eliminate duplicate segments. With the option of a resemblance index, the system could dynamically opt to execute chunk-level, segment resemblance-based deduplication to diminish upload volume. Although this step may slow throughput, it promises a significant reduction in the re-upload of previously stored chunks (RSD).

The decision to shift from a P-SCAIL to a PR-SCAIL-styled lookup could be based on various criteria, such as a predefined system-wide or client-specific threshold of upload volume, or a percentage of non-duplicate segments. Furthermore, the system might evaluate the number of segments resembling those previously stored to determine the appropriateness of chunk-level segment resemblance deduplication. This decision-making process could also leverage historical data, either from the server or individual client experiences. Future research will be crucial in developing sophisticated algorithms for adaptive threshold management, potentially incorporating machine learning techniques for anticipatory analysis.

7.5 Future Work: Adaptive Server-side Deduplicaton

In our exploration of deduplication techniques, we employed Sorted Chunk Indexing (SCI) for server-side deduplication within our hybrid two-phase deduplication framework. While SCI offers significant advantages in terms of low memory usage and low disk I/Os for large datasets, there is potential to further optimise server-side deduplication by adapting the deduplication strategy based on the characteristics of the data that reaches the server after client-side deduplication processing.

In typical backup scenarios, a substantial portion of redundant data is eliminated during client-side deduplication — especially in backups following the initial backup — the volume of data requiring server-side deduplication is significantly reduced. This observation opens up the possibility of exploring alternative server-side deduplication techniques that could be more efficient or better suited to certain workloads than SCI.

One potential approach is to reconsider the use of a traditional CFP index for server-side deduplication. Traditionally, these are avoided at large scales due to their high memory requirements, or the significant, random disk I/O associated with a disk-based full chunk index. However, with the reduced data volume resulting from effective client-side deduplication, the overhead associated with a CFP index might become manageable. The lower number of chunks to be deduplicated could reduce the number of disk I/O to below those required by SCI, leading to faster deduplication times.

Another avenue is the exploration of low-memory, fast, inexact server-side deduplication techniques. These techniques, which may use probabilistic data structures like Bloom Filters can quickly identify duplicate chunks with a small chance of false positives. While this may result in some duplicate chunks being stored, the additional storage overhead might be acceptable when balanced against the benefits of reduced deduplication time and resource usage. Moreover, the storage savings from metadata deduplication could offset the impact of storing some redundant chunks.

7.6 Conclusion

In this thesis, we investigated increasing the data capacity and efficiency in encrypted deduplication backup systems. We successfully integrated the lower system requirements and high-speed deduplication throughput of coarse granularity chunking with the high storage reductions of fine-grained deduplication. We developed several novel approaches that effectively extend and integrate these factors in SCAIL.

Once we had achieved these higher capacities, we found the need to increase the throughput of the server system to handle expanded data and client capacity, which we achieved by improving cache management and leveraging multiprocessor server architectures in P-SCAIL.

The development of SCAIL and PR-SCAIL led to a new area of research: reducing the volume of redundancy segment data (RSD) uploads generated by these designs. This led to an analysis of RSD and the implementation of a new segment resemblance technique introduced in PR-SCAIL.

Through these innovations, we have advanced the field of encrypted deduplication, offering scalable, efficient, and secure solutions to pressing challenges faced by modern data storage systems. Our work lays the groundwork for future research in the design of next-generation deduplication systems in secure environments.

Bibliography

[1] Atul Adya, William J Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R Douceur, Jon Howell, Jacob R Lorch, Marvin Theimer, and Roger P Wattenhofer. "Farsite: federated, available, and reliable storage for an incompletely trusted environment". In: *ACM SIGOPS Operating Systems Review* 36.SI (Dec. 2003), pp. 1–14.

[2] Bhavish Agarwal, Aditya Akella, Ashok Anand, Athula Balachandran, Pushkar V Chitnis, Chitra Muthukrishnan, Ramachandran Ramjee, and George Varghese. "Endre: An end-system redundancy elimination service for enterprises". In: *NSDI*. Vol. 10. 2010, pp. 419–432.

[3] Ashok Anand, Chitra Muthukrishnan, Aditya Akella, and Ramachandran Ramjee. "Redundancy in network traffic: findings and implications". In: *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*. SIGMETRICS '09. New York, NY, USA: Association for Computing Machinery, June 2009, pp. 37–48.

[4] Paul Anderson and Le Zhang. "Fast and Secure Laptop Backups with Encrypted De-duplication". In: *Research Gate* (2010).

[5] L Aronovich, R Asher, D Harnik, M Hirsch, S T Klein, and Y Toaff. "Similarity based deduplication with small data chunks". In: *Discrete applied mathematics* 212 (2016), pp. 10–22.

[6] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. *Provable data possession at untrusted stores*. 2007.

[7] *Azure Pricing – bandwidth*. en. `https://azure.microsoft.com/en-gb/pricing/details/bandwidth/`. Accessed: 2022-4-12.

[8] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. "Message-Locked Encryption and Secure Deduplication". In: *Advances in Cryptology – EUROCRYPT 2013*. Springer Berlin Heidelberg, 2013, pp. 296–312.

[9] D Bhagwat, K Eshghi, D D E Long, and M Lillibridge. "Extreme Binning: Scalable, parallel deduplication for chunk-based file backup". In: *2009 IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems*. Sept. 2009, pp. 1–9.

[10] Pramod Bhatotia, Rodrigo Rodrigues, and Akshat Verma. "Shredder: GPU-accelerated incremental storage and computation". In: *FAST*. Vol. 14. 2012, p. 14.

[11] Nikolaj Bjørner, Andreas Blass, and Yuri Gurevich. "Content-dependent chunking for differential compression, the local maximum approach". In: *Journal of Computer and System Sciences* 76.3 (May 2010), pp. 154–203.

[12] Deepak R Bobbarjung, Suresh Jagannathan, and Cezary Dubnicki. "Improving duplicate elimination in storage systems". In: *ACM Trans. Storage* 2.4 (Nov. 2006), pp. 424–448.

[13] Andrei Z Broder. "Identifying and Filtering Near-Duplicate Documents". In: *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*. COM '00. Berlin, Heidelberg: Springer-Verlag, 2000, pp. 1–10.

[14] Andrei Z Broder. "On the resemblance and containment of documents". In: *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*. 1997, pp. 21–29.

[15] Andrei Z Broder. "Some applications of Rabin's fingerprinting method". In: *Sequences II*. Springer, 1993, pp. 143–152.

[16] Josiah Carlson. *Redis in Action*. en. Simon and Schuster, June 2013.

[17] T C E Cheng and H G Kahlbacher. "A proof for the longest-job-first policy in one-machine scheduling". en. In: *Naval Research Logistics* 38.5 (Oct. 1991), pp. 715–720.

[18] Yann Collet. *xxHash: Extremely fast non-cryptographic hash algorithm*. `https://xxhash.com/`. Accessed: 2023-9-1. Nov. 2023.

[19] Landon P Cox, Christopher D Murray, and Brian D Noble. "Pastiche: making backup cheap and easy". In: *ACM SIGOPS Operating Systems Review* 36.SI (Dec. 2003), pp. 285–298.

[20] Girum Dagnaw, Ke Zhou, and Hua Wang. "SACRO : Solid state drive-assisted chunk caching for restore optimization". en. In: *Concurrency and computation: practice & experience* 35.18 (Aug. 2023).

[21] Dan Dobre, Paolo Viotti, and Marko Vukolić. "Hybris: Robust Hybrid Cloud Storage". In: *Proceedings of the ACM Symposium on Cloud Computing*. New York, NY, USA: ACM, Nov. 2014.

[22] Wei Dong, Fred Douglis, Kai Li, R Hugo Patterson, Sazzala Reddy, and Philip Shilane. "Tradeoffs in Scalable Data Routing for Deduplication Clusters". In: *FAST*. Vol. 11. 2011, pp. 15–29.

[23] J R Douceur, A Adya, W J Bolosky, P Simon, and M Theimer. "Reclaiming space from duplicate files in a serverless distributed file system". In: *Proceedings 22nd International Conference on Distributed Computing Systems*. July 2002, pp. 617–624.

[24] F Douglis, Deepti Bhardwaj, Hangwei Qian, and Philip Shilane. "Content-aware load balancing for distributed backup". In: *LiSA* (Dec. 2011), pp. 13–13.

[25] K Eshghi and Henry Hong Ki Tang. "A framework for analyzing and improving content-based chunking algorithms". In: *Hewlett-Packard Labs Technical Report TR* 30.2005 (2005).

[26] Steven D Feldman, Akshatha Bhat, Pierre LaBorde, Qing Yi, and Damain Dechev. "Effective use of non-blocking data structures in a deduplication application". In: *Proceedings of the 2013 companion publication for conference on Systems, programming, & applications: software for humanity*. SPLASH '13. New York, NY, USA: Association for Computing Machinery, Oct. 2013, pp. 133–142.

[27] Min Fu, Shujie Han, Patrick P C Lee, Dan Feng, Zuoning Chen, and Yu Xiao. "A simulation analysis of redundancy and reliability in primary storage deduplication". In: *IEEE transactions on computers. Institute of Electrical and Electronics Engineers* 67.9 (Sept. 2018), pp. 1259–1272.

[28] Yinjin Fu, Hong Jiang, and Nong Xiao. "A scalable inline cluster deduplication framework for big data protection". In: *Lecture Notes in Computer Science*. Lecture notes in computer science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 354–373.

[29] Yinjin Fu, Nong Xiao, Hong Jiang, Guyu Hu, and Weiwei Chen. "Application-Aware Big Data Deduplication in Cloud Environment". In: *IEEE Transactions on Cloud Computing* 7.4 (Oct. 2019), pp. 921–934.

[30] Abdullah Gharaibeh, Samer Al-Kiswany, Sathish Gopalakrishnan, and Matei Ripeanu. "A GPU accelerated storage system". In: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. HPDC '10. New York, NY, USA: Association for Computing Machinery, June 2010, pp. 167–178.

[31] Fanglu Guo and Petros Efstathopoulos. "Building a High-performance Deduplication System". In: *USENIX annual technical conference*. 2011.

[32] Guanxiong Ha, Hang Chen, Chunfu Jia, and Mingyue Li. "Threat Model and Defense Scheme for Side-Channel Attacks in Client-Side Deduplication". In: *Tsinghua science and technology* 28.1 (Feb. 2023), pp. 1–12.

[33] Shai Halevi, Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. "Proofs of ownership in remote storage systems". In: *Proceedings of the 18th ACM conference on Computer and communications security*. CCS '11. New York, NY, USA: Association for Computing Machinery, Oct. 2011, pp. 491–500.

[34] D Harnik, B Pinkas, and A Shulman-Peleg. "Side Channels in Cloud Services: Deduplication in Cloud Storage". In: *IEEE Security Privacy* 8.6 (Nov. 2010), pp. 40–47.

[35] O Heen, C Neumann, L Montalvo, and S Defrance. "Improving the Resistance to Side-Channel Attacks on Cloud Storage Services". In: *2012 5th International Conference on New Technologies, Mobility and Security (NTMS)*. May 2012, pp. 1–5.

[36] J Kaiser, T Süß, L Nagel, and A Brinkmann. "Sorted deduplication: How to process thousands of backup streams". In: *2016 32nd Symposium on Mass Storage Systems and Technologies (MSST)*. May 2016, pp. 1–14.

[37] Jürgen Kaiser, André Brinkmann, Tim Süß, and Dirk Meister. "Deriving and comparing deduplication techniques using a model-based classification". In: *Proceedings of the Tenth European Conference on Computer Systems*. EuroSys '15 Article 11. New York, NY, USA: Association for Computing Machinery, Apr. 2015, pp. 1–13.

[38] Sriram Keelveedhi, Mihir Bellare, and Thomas Ristenpart. "DupLESS: server-aided encryption for deduplicated storage". In: *Presented as part of the 22nd Usenix Security Symposium (Usenix) Security 13)*. 2013, pp. 179–194.

[39] Chulmin Kim, Ki-Woong Park, and Kyu Ho Park. "GHOST: GPGPU-offloaded high performance storage I/O deduplication for primary storage system". In:

Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores. PMAM '12. New York, NY, USA: Association for Computing Machinery, Feb. 2012, pp. 17–26.

[40] Roei Kisous, Ariel Kolikant, Abhinav Duggal, Sarai Sheinvald, and Gala Yadgar. "The what, The from, and The to: The Migration Games in Deduplicated Systems". In: *ACM Trans. Storage* (Sept. 2022).

[41] Samer Al-Kiswany, Abdullah Gharaibeh, Elizeu Santos-Neto, George Yuan, and Matei Ripeanu. "StoreGPU: Exploiting Graphics Processing Units to Accelerate Distributed Storage Systems". In: *Proceedings of the 17th International Symposium on High Performance Distributed Computing*. HPDC '08. New York, NY, USA: ACM, 2008, pp. 165–174.

[42] Oleg Kolosov, Gala Yadgar, S Maheshwari, and E Soljanin. "Benchmarking in the dark: On the absence of comprehensive edge datasets". In: *USENIX Workshop on Hot Topics in Edge Computing* (2020).

[43] Erik Kruus, Cristian Ungureanu, and Cezary Dubnicki. "Bimodal content defined chunking for backup streams". In: *Fast*. 2010, pp. 239–252.

[44] J Li, P P C Lee, Y Ren, and X Zhang. "Metadedup: Deduplicating Metadata in Encrypted Deduplication via Indirection". In: *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*. May 2019, pp. 269–281.

[45] Jingwei Li, Chuan Qin, Patrick P C Lee, and Xiaosong Zhang. *Information Leakage in Encrypted Deduplication via Frequency Analysis*. 2017.

[46] Mingqiang Li, Chuan Qin, Jingwei Li, and Patrick P C Lee. *CDStore: Toward Reliable, Secure, and Cost-Efficient Cloud Storage via Convergent Dispersal*. 2016.

[47] Pengfei Li, Yu Hua, Qin Cao, and Mingxuan Zhang. "Improving the restore performance via physical-locality middleware for backup systems". In: *Proceedings*

of the 21st International Middleware Conference. New York, NY, USA: ACM, Dec. 2020.

[48] Mark Lillibridge, Kave Eshghi, and Deepavali Bhagwat. "Improving restore speed for backup systems that use inline chunk-based deduplication". In: *11th {USENIX} Conference on File and Storage Technologies ({FAST} 13)*. 2013, pp. 183–197.

[49] Mark Lillibridge, Kave Eshghi, Deepavali Bhagwat, Vinay Deolalikar, Greg Trezis, and Peter Camble. "Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality". In: *7th USENIX Conference on File and Storage Technologies (FAST 09)*. Vol. 9. 2009, pp. 111–123.

[50] Lifang Lin, Yuhui Deng, Yi Zhou, and Yifeng Zhu. "InDe: An inline data deduplication approach via adaptive detection of valid container utilization". In: *ACM Trans. Storage* (Nov. 2022).

[51] Chuanyi Liu, Yibo Xue, Dapeng Ju, and Dongsheng Wang. "A Novel Optimization Method to Improve De-duplication Storage System Performance". In: *2009 15th International Conference on Parallel and Distributed Systems*. IEEE, Dec. 2009, pp. 228–235.

[52] Saiqin Long, Zhetao Li, Zihao Liu, Qingyong Deng, Sangyoon Oh, and Nobuyoshi Komuro. "A similarity clustering-based deduplication strategy in cloud storage systems". In: *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*. Dec. 2020, pp. 35–43.

[53] Guanlin Lu, Yu Jin, and David H C Du. "Frequency Based Chunking for Data De-Duplication". In: *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, Aug. 2010.

[54] Shengmei Luo, Guangyan Zhang, Chengwen Wu, Samee U Khan, and Keqin Li. "Boafft: Distributed Deduplication for Big Data Storage in the Cloud". In: *IEEE Transactions on Cloud Computing* 8.4 (2020), pp. 1199–1211.

[55] Udi Manber. "Finding Similar Files in a Large File System". In: *Proceedings of the Winter 1994 USENIX Technical Conference*. Vol. 94. 1994, pp. 1–10.

[56] 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. New York, NY, USA: ACM, 2008, pp. 12–17.

[57] Dirk Meister, Jürgen Kaiser, and André Brinkmann. "Block Locality Caching for Data Deduplication". In: *Proceedings of the 6th International Systems and Storage Conference*. SYSTOR '13. New York, NY, USA: ACM, 2013, 15:1–15:12.

[58] Dutch T Meyer and William J Bolosky. "A study of practical deduplication". In: *ACM Transactions on Storage* 7.4 (2012), pp. 1–20.

[59] J Min, D Yoon, and Y Won. "Efficient Deduplication Techniques for Modern Backup Operation". In: *IEEE transactions on computers. Institute of Electrical and Electronics Engineers* 60.6 (June 2011), pp. 824–840.

[60] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, and Ion Stoica. "Ray: A Distributed Framework for Emerging AI Applications". In: *arXiv [cs.DC]* (Dec. 2017).

[61] Athicha Muthitacharoen, Benjie Chen, and David Mazières. "A low-bandwidth network file system". In: *ACM SIGOPS Operating Systems Review* 35.5 (2001), pp. 174–187.

[62] Aviv Nachman, Sarai Sheinvald, Ariel Kolikant, and Gala Yadgar. "GoSeed: Optimal Seeding Plan for Deduplicated Storage". In: *ACM Trans. Storage* 17.3 (Aug. 2021), pp. 1–28.

[63] Sabuzima Nayak and Ripon Patgiri. "Dr. Hadoop: In search of a needle in a haystack". In: *Distributed Computing and Internet Technology*. Lecture notes in computer science. Cham: Springer International Publishing, 2019, pp. 99–107.

[64] Fan Ni, Xing Lin, and Song Jiang. "SS-CDC: a two-stage parallel content-defined chunking for deduplicating backup storage". In: *Proceedings of the 12th ACM International Conference on Systems and Storage*. 2019, pp. 86–96.

[65] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. "The log-structured merge-tree (LSM-tree)". In: *Acta Informatica* 33.4 (June 1996), pp. 351–385.

[66] *Overview of AWS Data Transfer Costs*. en. `https://aws.amazon.com/blogs/architecture/overview-of-data-transfer-costs-for-common-architectures/`. Accessed: 2022-4-12.

[67] P Puzio, R Molva, M Önen, and S Loureiro. "ClouDedup: Secure Deduplication with Encrypted Data for Cloud Storage". In: *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*. Vol. 1. Dec. 2013, pp. 363–370.

[68] Sean Quinlan and Sean Dorward. "Venti: A new approach to archival data storage". In: *Conference on File and Storage Technologies (FAST 02)*. usenix.org, 2002.

[69] M O Rabin. "Fingerprinting by random polynomials". In: *Technical report: NAVTRADEVCEN. Naval Training Device Center* (1981).

[70] Edward Richardson. "Enhancing Data Recovery in Deduplication Backup Systems: A Novel Rewriting Algorithm to Minimize Fragmentation Effects". In: *Journal of Computer Science and Software Applications* 4.2 (Mar. 2024), pp. 15–19.

[71] Bartłomiej Romański, Łukasz Heldt, Wojciech Kilian, Krzysztof Lichota, and Cezary Dubnicki. "Anchor-driven subchunk deduplication". In: *Proceedings of the 4th Annual International Conference on Systems and Storage*. SYSTOR '11 Article 16. New York, NY, USA: Association for Computing Machinery, May 2011, pp. 1–13.

[72] Ahmed El-Shimi, Ran Kalach, Ankit Kumar, Adi Ottean, Jin Li, and Sudipta Sengupta. "Primary data deduplication—large scale study and system design". In: *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*. 2012, pp. 285–296.

[73] Youngjoo Shin, Dongyoung Koo, and Junbeom Hur. "A Survey of Secure Data Deduplication Schemes for Cloud Storage Systems". In: *ACM Comput. Surv.* 49.4 (Jan. 2017), 74:1–74:38.

[74] Alex Spiridonov, Sahil Thaker, and Sourabh Patwardhan. *Sharing and bandwidth consumption in the low bandwidth file system*. Tech. rep. Tech. rep., Department of Computer Science, University of Texas at Austin, 2005.

[75] Jan Stanek, Alessandro Sorniotti, Elli Androulaki, and Lukas Kencl. "A Secure Data Deduplication Scheme for Cloud Storage". In: *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, 2014, pp. 99–118.

[76] Mark W Storer, Kevin Greenan, Darrell D E Long, and Ethan L Miller. "Secure data deduplication". In: *Proceedings of the 4th ACM international workshop on Storage security and survivability*. StorageSS '08. New York, NY, USA: Association for Computing Machinery, Oct. 2008, pp. 1–10.

[77] Zhen "jason" Sun, Geoff Kuenning, Sonam Mandal, Philip Shilane, Vasily Tarasov, Nong Xiao, and Erez Zadok. "Cluster and Single-Node Analysis of Long-Term Deduplication Patterns". In: *ACM Trans. Storage* 14.2 (May 2018), pp. 1–27.

[78] Xingpeng Tang and Jingwei Li. "Improving online restore performance of backup storage via historical file access pattern". en. In: *Communications in Computer and Information Science*. Communications in computer and information science. Singapore: Springer Nature Singapore, 2022, pp. 365–376.

[79] Vasily Tarasov, Amar Mudrankit, Will Buik, Philip Shilane, Geoff Kuenning, and Erez Zadok. "FSL-Dedup Traces (SNIA IOTTA Trace Set 5228)". In: *SNIA IOTTA Trace Repository*. Ed. by Geoff Kuenning. Storage Networking Industry Association, May 2016.

[80] D Teodosiu, Nikolaj S Bjørner, Y Gurevich, M Manasse, and Joe Porkka. "Optimizing file replication over limited-bandwidth networks using remote differential compression". In: *Microsoft Corp* (Nov. 2006).

[81] Grant Wallace, Fred Douglis, Hangwei Qian, Philip Shilane, Stephen Smaldone, Mark Chamness, and Windsor Hsu. "Characteristics of backup workloads in production systems". In: *FAST*. Vol. 12. 2012, pp. 4–4.

[82] Longxiang Wang, Xiaoshe Dong, Xingjun Zhang, Fuliang Guo, Yinfeng Wang, and Weifeng Gong. "A Logistic Based Mathematical Model to Optimize Duplicate Elimination Ratio in Content Defined Chunking Based Big Data Storage System". In: *Symmetry* 8.7 (July 2016), p. 69.

[83] Zooko Wilcox-O'Hearn and Brian Warner. "Tahoe: the least-authority filesystem". In: *Proceedings of the 4th ACM international workshop on Storage security and survivability*. StorageSS '08. New York, NY, USA: Association for Computing Machinery, Oct. 2008, pp. 21–26.

[84] J W J Williams. "Algorithm 232". In: *Communications of the ACM* (1964).

[85] Tony Wong, Smriti Thakkar, Kao-Feng Hsieh, Zachary Tom, Hetaben Saraiya, and Philip Shilane. "Dataset similarity detection for global deduplication in the

DD file system". In: *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, Apr. 2023, pp. 3322–3335.

[86] Shaoqiang Wu, Chunfu Jia, and Ding Wang. "UP-MLE: Efficient and practical updatable block-level message-locked encryption scheme based on update properties". In: *ICT Systems Security and Privacy Protection*. IFIP advances in information and communication technology. Cham: Springer International Publishing, 2022, pp. 251–269.

[87] Xiaotong Wu, Jiaquan Gao, Genlin Ji, Taotao Wu, Yuan Tian, and Najla Al-Nabhan. "A feature-based intelligent deduplication compression system with extreme resemblance detection". In: *Connection science* (Dec. 2020), pp. 1–29.

[88] W Xia, H Jiang, D Feng, F Douglis, P Shilane, Y Hua, M Fu, Y Zhang, and Y Zhou. "A Comprehensive Study of the Past, Present, and Future of Data Deduplication". In: *Proceedings of the IEEE* 104.9 (Sept. 2016), pp. 1681–1710.

[89] W Xia, H Jiang, D Feng, and Y Hua. "Similarity and Locality Based Indexing for High Performance Data Deduplication". In: *IEEE transactions on computers. Institute of Electrical and Electronics Engineers* 64.4 (Apr. 2015), pp. 1162–1176.

[90] W Xia, H Jiang, D Feng, and L Tian. "DARE: A Deduplication-Aware Resemblance Detection and Elimination Scheme for Data Reduction with Low Overheads". In: *IEEE transactions on computers. Institute of Electrical and Electronics Engineers* 65.6 (June 2016), pp. 1692–1705.

[91] Wen Xia, Dan Feng, Hong Jiang, Yucheng Zhang, Victor Chang, and Xiangyu Zou. "Accelerating content-defined-chunking based data deduplication by exploiting parallelism". In: *Future generations computer systems: FGCS* 98 (Sept. 2019), pp. 406–418.

[92] Wen Xia, Hong Jiang, Dan Feng, and Yu Hua. "SiLo: A Similarity-Locality based Near-Exact Deduplication Scheme with Low RAM Overhead and High Throughput". In: *USENIX Annual Technical Conference*. 2011, pp. 26–30.

[93] Wen Xia, Hong Jiang, Dan Feng, and Lei Tian. "Accelerating data deduplication by exploiting pipelining and parallelism with multicore or manycore processors". In: *Proc. 10th USENIX Conf. File Storage Technol*. 2012, pp. 1–2.

[94] Wen Xia, Hong Jiang, Dan Feng, Lei Tian, Min Fu, and Zhongtao Wang. "P-dedupe: Exploiting parallelism in data deduplication system". In: *2012 IEEE Seventh International Conference on Networking, Architecture, and Storage*. IEEE, June 2012.

[95] Wen Xia, Hong Jiang, Dan Feng, Lei Tian, Min Fu, and Yukun Zhou. "Ddelta: A deduplication-inspired fast delta compression approach". In: *Performance Evaluation* 79 (Sept. 2014), pp. 258–272.

[96] Zhen Xu and Wenbo Zhang. "QuickCDC: A Quick Content Defined Chunking Algorithm Based on Jumping and Dynamically Adjusting Mask Bits". In: *2021 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom)*. Sept. 2021, pp. 288–299.

[97] Zuoru Yang, Jingwei Li, and Patrick P C Lee. "Secure and Lightweight Deduplicated Storage via Shielded Deduplication-Before-Encryption". In: *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 2022, pp. 37–52.

[98] Zuoru Yang, Jingwei Li, Yanjing Ren, and Patrick P C Lee. "Tunable Encrypted Deduplication with Attack-resilient Key Management". In: *ACM Trans. Storage* 18.4 (Nov. 2022), pp. 1–38.

[99] Wenbin Yao, Mengyao Hao, Yingying Hou, and Xiaoyong Li. "FASR: An efficient feature-aware deduplication method in distributed storage systems". In: *IEEE Access: Practical Innovations, Open Solutions* 10 (2022), pp. 15311–15321.

[100] Chuanshuai Yu, Chengwei Zhang, Yiping Mao, and Fulu Li. "Leap-based Content Defined Chunking — Theory and Implementation". In: *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, May 2015, pp. 1–12.

[101] C Zhang, D Qi, W Li, and J Guo. "Function of Content Defined Chunking Algorithms in Incremental Synchronization". In: *IEEE Access* 8 (2020), pp. 5316–5330.

[102] Changjian Zhang, Deyu Qi, Wenlin Li, Wenhao Huang, and Xinyang Wang. "SimpleSync: A parallel delta synchronization method based on Flink". en. In: *Concurrency and Computation: Practice & Experience* 33.20 (Oct. 2021).

[103] Datong Zhang, Yuhui Deng, Yi Zhou, Yifeng Zhu, and Xiao Qin. "Improving the Performance of Deduplication-Based Backup Systems via Container Utilization Based Hot Fingerprint Entry Distilling". In: *ACM Trans. Storage* 17.4 (Oct. 2021), pp. 1–23.

[104] P Zhang, P Huang, X He, H Wang, L Yan, and K Zhou. "RMD: A Resemblance and Mergence Based Approach for High Performance Deduplication". In: *2016 45th International Conference on Parallel Processing (ICPP)*. Aug. 2016, pp. 536–541.

[105] Y Zhang, H Jiang, D Feng, W Xia, M Fu, F Huang, and Y Zhou. "AE: An Asymmetric Extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication". In: *2015 IEEE Conference on Computer Communications (INFOCOM)*. Apr. 2015, pp. 1337–1345.

[106] Yucheng Zhang, Wen Xia, Dan Feng, Hong Jiang, Yu Hua, and Qiang Wang. "Finesse: fine-grained feature locality based fast resemblance detection for post-deduplication delta compression". In: *17th {USENIX} Conference on File and Storage Technologies ({FAST} 19)*. 2019, pp. 121–128.

[107] Zhen Sun, N Xiao, F Liu, and Y Fu. "DS-Dedupe: A scalable, low network overhead data routing algorithm for inline cluster deduplication system". In: *2014 International Conference on Computing, Networking and Communications (ICNC)*. Feb. 2014, pp. 895–899.

[108] B Zhou and J Wen. "Hysteresis Re-chunking Based Metadata Harnessing Deduplication of Disk Images". In: *2013 42nd International Conference on Parallel Processing*. Oct. 2013, pp. 389–398.

[109] Benjamin Zhu, Kai Li, and R Hugo Patterson. "Avoiding the Disk Bottleneck in the Data Domain Deduplication File System". In: *6th USENIX Conference on File and Storage Technologies (FAST 08)*. Vol. 8. 2008, pp. 1–14.

[110] Xiangyu Zou, Cai Deng, Wen Xia, Philip Shilane, Haoliang Tan, Haijun Zhang, and Xuan Wang. "Odess: Speeding up Resemblance Detection for Redundancy Elimination by Fast Content-Defined Sampling". In: *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. ieeexplore.ieee.org, Apr. 2021, pp. 480–491.