Summer 5-30-2012

A Scalable Inline Cluster Deduplication Framework for Big Data Protection

Yinjin Fu
University of Nebraska-Lincoln & National University of Defense Technology, China, yfu@cse.unl.edu

Hong Jiang
University of Nebraska-Lincoln, jiang@cse.unl.edu

Nong Xiao
National University of Defense Technology, China, nongxiao@nudt.edu.cn

A Scalable Inline Cluster Deduplication Framework for Big Data Protection

Yinjin Fu[1, 2], Hong Jiang[2], Nong Xiao[1]

[1] State Key Laboratory of High Performance Computing, School of Computer, National University of Defense Technology, China
yinjinfu@gmail.com, nongxiao@nudt.edu.cn
[2] Department of Computer Science and Engineering, University of Nebraska-Lincoln, USA
{jiang,yfu}@cse.unl.edu

Abstract. Cluster deduplication has become a widely deployed technology in data protection services for Big Data to satisfy the requirements of service level agreement (SLA). However, it remains a great challenge for cluster deduplication to strike a sensible tradeoff between the conflicting goals of scalable deduplication throughput and high duplicate elimination ratio in cluster systems with low-end individual secondary storage nodes. We propose Σ-Dedupe, a scalable inline cluster deduplication framework, as a middleware deployable in cloud data centers, to meet this challenge by exploiting data similarity and locality to optimize cluster deduplication in inter-node and intra-node scenarios, respectively. Governed by a similarity-based stateful data routing scheme, Σ-Dedupe assigns similar data to the same backup server at the super-chunk granularity using a handprinting technique to maintain high cluster-deduplication efficiency without cross-node deduplication, and balances the workload of servers from backup clients. Meanwhile, Σ-Dedupe builds a similarity index over the traditional locality-preserved caching design to alleviate the chunk index-lookup bottleneck in each node. Extensive evaluation of our Σ-Dedupe prototype against state-of-the-art schemes, driven by real-world datasets, demonstrates that Σ-Dedupe achieves a cluster-wide duplicate elimination ratio almost as high as the high-overhead and poorly scalable traditional stateful routing scheme but at an overhead only slightly higher than that of the scalable but low duplicate-elimination-ratio stateless routing approaches.

Keywords: Big Data protection, cluster deduplication, data routing, super-chunk, handprinting, similarity index, load balance

1 Introduction

The explosive growth of data in volume and complexity in our digital universe is occurring at a record rate, growing by almost 9 times to 7 zettabytes per year in past five years and more than 44 fold to 35 zettabytes expected in the next ten years, according to a recent IDC study [1]. Enterprises are awash in digital data, easily amassing petabytes and even exabytes of information, and the risk of data loss escalates due

to the growing complexity of data management in Big Data. No matter how the data is lost, it is costly for an enterprise. In response to an IDC research survey [2], almost half of the companies, of all sizes, reported that the total impact of financial loss per event of data breach was over $100,000, while 8.5% of the enterprises reported financial loss of over $1 million. One of the best protection strategies against threats is to backup data locally or remotely. The frequency, type and retention of backups vary for different kinds of data, but it is common for the secondary storage in enterprises to hold tens of times more data than the primary storage, and more data stored and moved for disaster recovery [3,17]. The sprawling of backup storage systems not only consumes more data storage space, power and cooling in data centers, it also adds significant administration time and increases operational complexity and risk of human error. Meanwhile, to satisfy the high velocity requirements in modern storage systems, memory becomes the new disk, and disk becomes the new tape. Managing the data deluge under the changes in storage media to meet the SLA requirements becomes an increasingly critical challenge for Big Data protection.

Data deduplication, a specialized data reduction technique widely deployed in disk-based backup systems, partitions large data objects into smaller parts, called chunks, and represents and replaces these chunks by their fingerprints (i.e., generally a cryptographic hash of the chunk data) for the purpose of improving communication and storage efficiency by eliminating data redundancy in various application datasets. IDC data shows that nearly 75% of our digital world is a copy [4], and over 90% data is duplicated in backup datasets [5]. Source inline data deduplication is favored in industry and academia, because it can immediately identify and eliminate duplicates in datasets at the source of data generation and hence significantly reduce physical storage capacity requirements and save network bandwidth during data transfer. To satisfy scalable capacity and performance requirements in Big Data protection, cluster deduplication [6,7,8,9,11,12] has been proposed to provide high deduplication throughput in massive backup data. It includes inter-node data assignment from backup clients to multiple deduplication nodes by a data routing scheme, and independent intra-node deduplication in individual nodes. Unfortunately, chunk-based inline cluster deduplication at large scales faces challenges in both inter-node and intra-node scenarios. First, for the inter-node scenario, it achieves a good balance between capacity saving and performance scalability at the cost of a low duplicate elimination ratio caused by *deduplication node information island*. This means that deduplication is only performed within individual servers due to overhead considerations, which leaves cross-node redundancy untouched. Thus, data routing, a technique to concentrate data redundancy within individual nodes, reduce cross-node redundancy and balance load, becomes a key issue in the cluster deduplication design. Second, for the intra-node scenario, it suffers from *the disk chunk index lookup bottleneck*. That is, the chunk index of a large dataset, which maps each chunk's fingerprint to where that chunk is stored on disk in order to identify the replicated data, is generally too big to fit into the limited memory of a deduplication server and causes the parallel deduplication performance of multiple data streams from backup clients to degrade significantly due to the frequent and random disk I/Os to look up the chunk index.

There are several existing solutions that aim to tackle these two challenges of cluster deduplication by exploiting data similarity or locality. Locality based approaches, such as the stateless routing and stateful routing schemes [6], exploit locality in backup data streams to optimize cluster deduplication. These schemes distribute data across deduplication servers at coarse granularity to achieve scalable deduplication throughput across the nodes, while deduplicate data at fine granularity in individual servers for high deduplication effectiveness (i.e., duplicate elimination ratio) in each node. However, to achieve high cluster deduplication effectiveness, it requires very high communication overhead to route similar data to the same node. Similarity based methods leverage data similarity to distribute data among deduplication nodes and reduce RAM usage in individual nodes. While these methods can easily find the node with highest similarity by extracting similarity features in the backup data streams, they often fail to obtain high deduplication effectiveness in individual deduplication servers. Extreme Binning [8] is a well-know example of similarity-based approaches that exploit file similarity in backup cluster systems. A more recent study, called SiLo [18], exploits both locality and similarity in backup streams to achieve a near-exact deduplication but at a RAM cost that is much lower than locality-only or similarity-only based methods. However, SiLo only addresses the intra-node challenge of single deduplication server. Inspired by SiLo, we aim to exploit data similarity and locality to strike a sensible tradeoff between the conflicting goals of high deduplication effectiveness and high performance scalability for cluster deduplication.

In this paper, we propose Σ-Dedupe, a scalable source inline cluster deduplication framework to overcome the aforementioned shortcomings of existing state-of-the-art cluster deduplication schemes, as a middleware deployable in data centers and cloud storage environments, to support Big Data protection. The main idea behind Σ-Dedupe is to optimize cluster deduplication by exploiting data similarity and locality in backup data streams. More specifically, to capture and maintain data locality in individual deduplication server, we adopt the notion of super-chunk [6], which represents consecutive smaller chunks of data, as a unit for data routing that assigns super-chunks to nodes and then performs deduplication at each node independently and in parallel. We extract the super-chunk feature by using handprinting technique, a new application of deterministic sampling, to detect resemblance among super-chunks. According to the super-chunk handprint, we choose a small subset of nodes in the deduplication sever cluster as route candidates, and send the handprint to the selected nodes to calculate the resemblance of the super-chunk by comparing its handprint with the handprints of previously stored super-chunks. To balance the workload among the nodes, we discount the super-chunk resemblances in the candidate nodes by storage usage in these nodes, and then find the candidate node with the highest discounted resemblance as the target node to route the super-chunk. After the target node selection, all the fingerprints of those chunks belonging to the super-chunk are sent to the target node to determine whether these chunks are duplicate or unique. Finally, the backup client only needs to send the unique chunks of the super-chunk to the target node. To reduce the overhead of resemblance detection in each node, we build a similarity index to store the handprints of the stored super-chunks in each

node, which also helps to alleviate the chunk disk index bottleneck for the deduplication processes in individual nodes by combining it with the conventional container-management based locality-preserved caching scheme [3].

The proposed Σ-Dedupe cluster deduplication system has the following salient features that distinguish it from the existing state-of-the-art cluster deduplication schemes:

- Σ-Dedupe exploits data similarity and locality by applying a handprinting technique at the super-chunk level to direct data routing from backup clients to deduplication server nodes to achieve a good tradeoff between the conflicting goals of high cluster deduplication effectiveness and highly scalable deduplication throughput.
- Σ-Dedupe builds a similarity index over the traditional container-based locality-preserved caching scheme to alleviate the chunk disk index lookup bottleneck for the deduplication process in each deduplication server node.
- Evaluation results from our prototype implementation of Σ-Dedupe show that it consistently and significantly outperforms the existing state-of-the-art cluster deduplication schemes in cluster deduplication efficiency by achieving high cluster deduplication effectiveness with balanced storage usage across the nodes and high parallel deduplication throughput at a low inter-node communication overhead. In addition, it maintains a high single-node deduplication performance with low RAM usage.

The rest of the paper is structured as follows. Section 2 presents the necessary background and related work to motivate the design of the Σ-Dedupe framework. Section 3 describes the architecture of our cluster deduplication system, the similarity based data routing algorithm and similarity-index based optimization technique. Section 4 evaluates the Σ-Dedupe prototype with real-world datasets, and Section 5 draws conclusions.

2 Background and Motivation

In this section, we first provide the necessary background and related work for our research by introducing the cluster deduplication techniques, and then present data similarity analysis based on a handprinting technique to motivate our research in the scalable inline cluster deduplication for Big Data protection.

2.1 Cluster Deduplication Techniques

Deduplication is both I/O intensive and compute intensive. Its process can be divided into four steps: data chunking, chunk fingerprint calculation, chunk index lookup, and unique data store. Source deduplication is a popular scheme that performs the first two steps of the deduplication process at the client side and decides whether a chunk is a duplicate before data transfer to save network bandwidth by avoiding the transfer

of redundant data, which differs from target deduplication that performs all deduplication steps at the target side. To immediately identify and eliminate data redundancy, inline deduplication is a process that performs deduplication on the traditional data I/O path with some impact on I/O performance. The throughput and capacity limitations of single-node deduplication have led to the development of cluster deduplication to provide high deduplication throughput in massive backup data. In our scheme, in order to shorten the backup window and improve the system scalability by reducing data transfer over the network, we choose source inline cluster deduplication to optimize the backup data storage in large-scale storage systems. However, in cluster deduplication design, in addition to the design challenge of the traditional chunk index structure in single-node deduplication, the design of data routing for the assignment of data to deduplication nodes has become a difficult challenge in achieving high global duplicate elimination ratio and scalable performance with balanced workload across the deduplication nodes.

Many existing cluster deduplication schemes, such as EMC Data Domain's global deduplication array [24], IBM's ProtecTier [23], and SEPATON's S2100-ES2 [25], are designed to work well in small clusters. But using these technologies to scale to thousands of nodes in cloud datacenters would most likely fail due to some of their shortcomings in terms of cluster-wide deduplication ratio, single-node throughput, data skew, and communication overhead. Hence, the design of inter-node data routing scheme and intra-node deduplication in large-scale cluster deduplication has become increasingly critical in recent years.

HYDRAstor [9] performs deduplication at a large-chunk (64KB) granularity without data sharing among the nodes, and distributes data at the chunk level using distributed hash table (DHT). It adopts large chunk size to achieve high deduplication throughput because of better data locality and less metadata overhead in large data chunks. Nevertheless, 64KB is still too limited to capture and preserve sufficient amount of locality for cluster deduplication purposes. While its chunk-level DHT based data routing is effective in lowering communication overhead and avoiding data sharing across the deduplication nodes, the intra-node local duplicate elimination ratio is reduced due to the large chunk size that tends to evade redundancy detection.

EMC's super-chunk based data routing [6] exploits data locality to direct data routing at the super-chunk level. It can route data evenly at the coarse-grained super-chunk level to preserve data locality and keep load balanced for scalable deduplication performance, and perform a fine-grained chunk-level deduplication to achieve high deduplication effectiveness for intra-node local deduplication. Depending on whether the information on previously stored data is used, super-chunk based data routing can be divided into stateless routing, which is oblivious of the state of the stored data, and stateful routing, which makes routing decision based on the state of the stored data. Stateless routing is also based on DHT with low overhead and can effectively balance workload in small clusters, but suffers from severe load imbalance in large clusters. Stateful routing is designed for large clusters to achieve high global deduplication effectiveness by effectively detecting cross-node data redundancy with the state information, but at the cost of very high system overhead required to route similar data to the same node.

Extreme Binning [8] is a file-similarity based cluster deduplication scheme. It can easily route similar data to the same deduplication node by extracting similarity characteristics in backup streams, but often suffers from low duplicate elimination ratio when data streams lack detectable similarity. It also has high data skew for the stateless routing due to the skew of file size distribution as studied in [13] and [17]. Similar to Extreme Binning, a new file-similarity based data routing scheme is proposed by Symantec [22] recently, but only a rough design is presented.

Table 1 compares some of the typical and representative cluster deduplication schemes, as discussed above. In relation to these existing approaches, our Σ-Dedupe is most relevant to Extreme Binning, and EMC's super-chunk based data routing (Stateless and Stateful). It aims to overcome many of the weaknesses described about these schemes. Comparing with Extreme Binning, Σ-Dedupe performs stateful data routing with a strong ability to discover similarity at the super-chunk level instead of the file level to enhance cluster-wide deduplication ratio and reduce data skew. Similar to EMC's super-chunk based data routing, Σ-Dedupe can preserve data locality at the super-chunk granularity, but is different from the former in that it exploits strong similarity at the super-chunk level to route data by a handprinting technique and only performs local stateful routing to keep load balanced and lower system overhead.

Table 1. Comparison of key features among representative cluster deduplication schemes

Cluster Deduplication Scheme	Routing Granularity	Deduplication Ratio	Throughput	Data Skew	Overhead
NEC HydraStor	Chunk	Medium	Low	Low	Low
Extreme Binning	File	Medium	High	Medium	Low
EMC Stateless	Super-chunk	Medium	High	Medium	Low
EMC Stateful	Super-chunk	High	Low	Low	High
Σ-Dedupe	Super-chunk	High	High	Low	Low

2.2 Super-chunk Resemblance Analysis

In the hash based deduplication schemes, cryptographic hash functions, such as the MD5 and SHA families of functions, are used for calculating chunk fingerprints due to their very low probability of hash collisions that renders data loss extremely unlikely. Assume that two different data chunks have different fingerprint values; we use the *Jaccard index* [14] as a measure of super-chunk resemblance. Let h be a cryptographic hash function, $h(S)$ denote the set of chunk fingerprints generated by h on super-chunk S. Hence, for any two super-chunks S_1 and S_2 with almost the same average chunk size, we can define their resemblance measure $r(S_1, S_2)$ according to the Jaccard index as expressed in Eq (1).

$$r(S_1,S_2) \triangleq \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} \approx \frac{|h(S_1) \cap h(S_2)|}{|h(S_1) \cup h(S_2)|} \quad (1)$$

Our similarity based data routing scheme depends on the creative feature selection on super-chunks by a handprinting technique. The selection method is based on a

generalization of Broder's theorem [15]. Before we discuss the theorem, let's first introduce the min-wise independent hash functions.

Definition 1. A family of hash functions $H = \{h_i: [n] \rightarrow [n]\}$ (where $[n]=\{0, 1, \ldots, n-1\}$) is called *min-wise independent* if for any $X \subset [n]$ and $x \in X$, it can be formally stated as in Eq. (2), where $\Pr_{h \in H}$ denotes the probability space obtained by choosing h uniformly at random from H.

$$\Pr_{h \in H}(\min\{h(X)\} = h(x)) = \frac{1}{|X|} \tag{2}$$

As the truly min-wise independent hash functions are hard to implement, practical systems only use hash functions that approximate min-wise independence, such as functions of the MD/SHA family cryptographic hash functions.

Theorem 1. (Broder's Theorem): For any two super-chunks S_1 and S_2, with $h(S_1)$ and $h(S_2)$ being the corresponding sets of the chunk fingerprints of the two super-chunks, respectively, where h is a hash function that is selected uniformly and at random from a min-wise independent family of cryptographic hash functions. Then Eq. (3) is established.

$$\Pr(\min\{h(S_1)\} = \min\{h(S_2)\}) = r(S_1, S_2) \tag{3}$$

Considering that h is a min-wise independent cryptographic hash function, for any $x \in S_1$, $y \in S_2$, the probability of x equaling y is the Jaccard index based resemblance $r(S_1, S_2)$, then we have a result as expressed in Eq. (4). Since there are $|S_1|$ choices for x and $|S_2|$ choices for y, Eq. (3) in the above theorem is established.

$$\Pr(\min\{h(S_1)\} = h(x) \wedge \min\{h(S_2)\} = h(y) \wedge h(x) = h(y)) = \frac{r(S_1, S_2)}{|S_1| \bullet |S_2|} \tag{4}$$

We consider a generalization of Broder's Theorem, given in [10], for any two super-chunks S_1 and S_2, and then we have a conclusion expressed in Eq. (5), where $\min_k$ denotes the k smallest elements in a set. It means that we can use the k smallest chunk fingerprints as representative fingerprints of a super-chunk to construct a *handprint* for it to find more similarity in datasets. With k being the handprint size, two super-chunks will more likely be found similar.

$$\begin{aligned} &\Pr(\min_k\{h(S_1)\} \cap \min_k\{h(S_2)\} \neq \varnothing) \\ &= 1 - \Pr(\min_k\{h(S_1)\} \cap \min_k\{h(S_2)\} = \varnothing) \\ &\geq 1 - (1 - r(S_1, S_2))^k \geq r(S_1, S_2) \end{aligned} \tag{5}$$

We evaluate the effectiveness of handprinting on super-chunk resemblance detection in the first 8MB super-chunks of four pair-wise files with different application types, including Linux 2.6.7 versus 2.6.8 kernel packages, and pair-wise versions of PPT, DOC and HTML files. We actually use the Two-Threshold Two-Divisor (TTTD) chunking algorithm [16] to subdivide the super-chunk into small chunks with 1KB, 2KB, 4KB and 32KB as minimum threshold, minor mean, major mean and maximum threshold of chunk size, respectively. TTTD is a variant of the basic content defined chunking (CDC) algorithm that leads to superior deduplication. We can calculate the real resemblance value based on the Jaccard index by the whole chunk fingerprint comparison on each pair of super-chunks, and estimate the resemblance by comparing representative fingerprints in handprint comparison with different

handprint sizes. The estimated resemblance, as shown in Figure 1 as a function of the handprint size, approaches the real resemblance value as the handprint size increases. An evaluation of Figure 1 suggests that a reasonable handprint size can be chosen in the range from 4 to 64 representative fingerprints. Comparing with the conventional schemes that only use a single representative fingerprint (when handprint size equals to 1), our handprinting method can find more similarity for file pairs with poor similarity (with a resemblance value of less than 0.5), such as the two PPT versions and the pair of HTML versions.

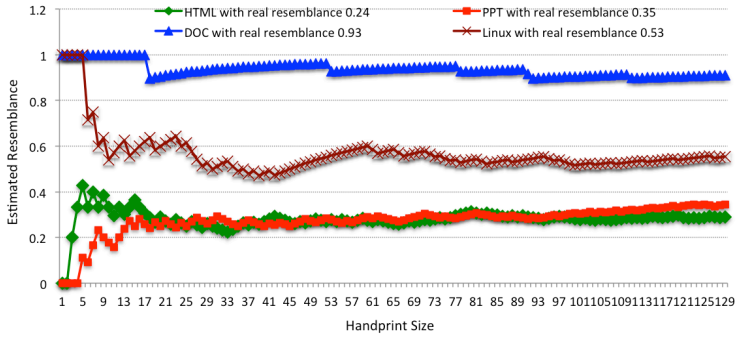


Fig. 1. The effect of handprinting resemblance detection

3 Σ-Dedupe Design

In this section, we present the design of the Σ-Dedupe cluster deduplication system. Besides the high throughput requirement in individual deduplication nodes, any cluster deduplication system must support scalable performance without significantly sacrificing capacity saving. We use the following design principles to govern our design for system architecture and data routing scheme:

- **Throughput:** The cluster deduplication throughput should scale with the number of nodes by parallel deduplication across the cluster nodes. Deduplication nodes should perform near-raw-disk data backup throughput by eliminating index lookup bottleneck, implying that our scheme must optimize for cache locality even with some but acceptable penalty on capacity saving.
- **Scalability:** The cluster deduplication system should easily scale out to handle massive data volumes with balanced workload among deduplication nodes, implying that our design must not only optimize the intra-node throughput by capturing and preserving high locality, but also reduce inter-node communication overhead for data routing by exploiting data similarity.
- **Capacity:** In backup data streams, similar data should be forwarded to the same deduplication node to achieve high duplicate elimination ratio. And capacity usage should be balanced across nodes to support high scalability and simplified system management. If system resources are scarce, deduplication effectiveness can be sacrificed to improve the system performance.

To achieve high deduplication throughput and good scalability with negligible capacity loss, we design a scalable inline cluster deduplication framework in this section. In what follows, we first show the architecture of our inline cluster deduplication system. Then we present our similarity based data routing algorithm to achieve scalable performance with high deduplication efficiency. This is followed by the description of the similarity index based lookup optimization for high deduplication throughput in deduplication nodes.

3.1 System Overview

The architecture of our cluster deduplication system is shown in Figure 2. It consists of three main components: backup clients, deduplication server cluster and director.

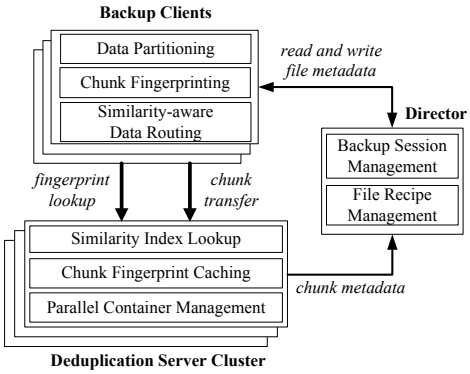


Fig. 2. Σ-Dedupe architectural overview

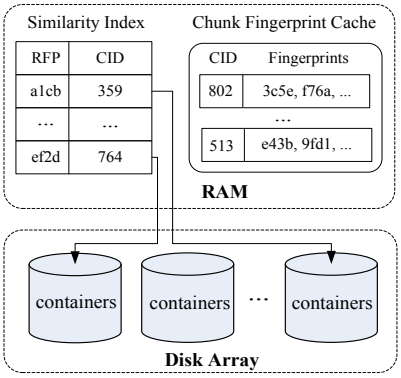


Fig. 3. Data structures in deduplication server

Backup clients. There are three main functional modules in a backup client: data partitioning, chunk fingerprinting and data routing. The backup client component backs up and restores data streams, performs data chunking with fixed or variable chunk size and super-chunk grouping in the data partitioning module for each data stream, and calculates chunk fingerprints by a collision-resistant hash function, like SHA-1 or MD5, then selects a deduplication node for the routing of each super-chunk by the data routing scheme. To improve cluster system scalability by saving the network transfer bandwidth during data backup, the backup clients determine whether a chunk is duplicate or not by batching chunk fingerprint query in the deduplication node at the super-chunk level before data chunk transfer, and only the unique data chunks are transferred over the network.

Deduplication server cluster. The deduplication server component consists of three important functional modules: similarity index lookup, chunk index cache management and parallel container management. It implements the key deduplication and backup management logic, including returning the results of similarity index lookup for data routing, buffering the recent hot chunk fingerprints in chunk index cache to speedup the process of identifying duplicate chunks and storing the unique chunks in larger units, called containers, in parallel.

Director. It is responsible for keeping track of files on the deduplication server, and managing file information to support data backup and restore. It consists of backup session management and file recipe management. The backup session management module groups files belonging to the same backup session of the same client, and file recipe management module keeps the mapping from files to chunk fingerprints and all other information required to reconstruct the file. All backup-session-level and file-level metadata are maintained in the director.

3.2 Similarity based data routing algorithm

As a new contribution of this paper, we present the similarity based data routing algorithm. It is a stateful data routing scheme motivated by our super-chunk resemblance analysis in Section 2. It routes similar data to the same deduplication node by looking up storage status information in only one or a small number of nodes, and achieves near-global capacity load balance without high system overhead. In the data partitioning module, a segment of the data stream is first divided it into n small chunks, that are grouped into a super-chunk S. Then, all the chunk fingerprints $\{fp_1, fp_2, \ldots, fp_n\}$ are calculated by a cryptographic hash function in the chunk fingerprinting module. The data routing algorithm, shown below, performs in the data routing module of backup clients.

Algorithm 1. Similarity based stateful data routing

Input: a chunk fingerprint list of super-chunk S, $\{fp_1, fp_2, \ldots, fp_n\}$
Output: a target node ID, i

1. Select the k smallest chunk fingerprints $\{rfp_1, rfp_2, \ldots, rfp_k\}$ as a handprint for the super-chunk S by sorting the chunk fingerprint list $\{fp_1, fp_2, \ldots, fp_n\}$, and sent the handprint to candidate nodes with IDs $\{rfp_1 \bmod N, rfp_2 \bmod N, \ldots, rfp_k \bmod N\}$ in the deduplicaton server cluster with N nodes;
2. In deduplication server cluster, obtain the count of the existing representative fingerprints of the super-chunk in the candidate nodes by comparing the representative fingerprints of the previously stored super-chunks in the similarity index. The returned k count values, one for each of the k candidate nodes, are denoted as $\{r_1, r_2, \ldots, r_k\}$, which are directly corresponding to the resemblances of S in these nodes;
3. Balance the capacity load in the candidate nodes, by discounting the resemblance value by the relative storage usage that is a node storage usage value divided by the average storage usage value, and the discounted resemblance values in the k candidate nodes, are denoted as $\{w_1, w_2, \ldots, w_k\}$;
4. Choose the deduplication server node with ID i that satisfies $r_i/w_i = \max\{r_1/w_1, r_2/w_2, \ldots, r_k/w_k\}$ as the target node.

Our similarity based data routing scheme can achieve local load balance for the k candidate nodes by adaptively choosing deduplication server node. We now prove that the global load balance can be approached by virtue of the universal distribution of randomly generated handprints by cryptographic hash functions, in Theorem 2.

Theorem 2. If each super-chunk handprint includes k fingerprints, and a local load balancing scheme is considered for the k candidate nodes with a mapping based on a modulo operation, then loads on all deduplication nodes can approach to the global load balance.

Proof. We prove by contradiction. Assume that the deduplication cluster consists of N nodes, $N \gg k$. First, we assume that our proposition is false. It means that there are at least two capacity load levels in the deduplication cluster without global load balance. We can divide all nodes into two groups by load level, denoted $\{H_1,\ldots, H_i\}$ for high load level and $\{L_1,\ldots, L_j\}$ for low load level, where $i + j = N$. For any super-chunk, the fingerprints in its handprint all map to the high-load group or low-load group, which means that all the datasets can be divided into two groups with the same cryptographic hash function. We know that this conclusion contradicts with the universal distribution property of cryptographic hash functions. Hence, our proposition must be true. We will further evaluate the theorem by experiments on real datasets in Section 4.

3.3 Similarity index based deduplication optimization

We outline the salient features of the key data structures designed for the deduplication server architecture. As shown in Figure 3, to support high deduplication throughput with low system overhead, a chunk fingerprint cache and two key data structures, similarity index and container, are introduced in our design.

Similarity index is a hash-table based memory data structure, with each of its entry containing a mapping between a representative fingerprint (RFP) in a super-chunk handprint and the container ID (CID) where it is stored. To support concurrent lookup operations in similarity index by multiple data streams on multicore deduplication nodes, we adopt a parallel similarity index lookup design and control the synchronization scheme by allocating a lock per hash bucket or for a constant number of consecutive hash buckets.

Container is a self-describing data structure stored in disk to preserve locality, similar to the one described in [3], that includes a data section to store data chunks and a metadata section to store their metadata information, such as chunk fingerprint, offset and length. Our deduplication server design supports parallel container management to allocate, deallocate, read, write and reliably store containers in parallel. For parallel data store, a dedicated open container is maintained for each coming data stream, and a new one is opened up when the container fills up. All disk accesses are performed at the granularity of a container.

Besides the two important data structures, the chunk fingerprint cache also plays a key role in deduplication performance improvement. It keeps the chunk fingerprints of recently accessed containers in RAM. Once a representative fingerprint is matched by a lookup request in the similarity index, all the chunk fingerprints belonging to the mapped container are prefetched into the chunk fingerprint cache to speedup chunk fingerprint lookup. The chunk fingerprint cache is a key-value structure, and it is constructed by a doubly linked list indexed by a hash table. When the cache is full, fin-

gerprints of those containers that are ineffective in accelerating chunk fingerprint lookup are replaced to make room for future prefetching and caching. A reasonable cache replacement policy is Least-Recently-Used (LRU) on cached chunk fingerprints. To support high deduplication effectiveness, we also maintain a traditional hash-table based chunk fingerprint index on disk to support further comparison after in-cache fingerprint lookup fails, but we consider it as a relatively rare occurrence.

To backup a super-chunk, after selecting the target node by our data routing algorithm, we resort to looking up the representative fingerprints in the similarity index. When a representative fingerprint is matched, we find the mapped container in the chunk fingerprint cache. If the container is already cached, we compare the fingerprints in the super-chunk with all the chunk fingerprints in the corresponding container; otherwise, we prefetch the fingerprints of that container from its metadata section before further comparison. After the search in all containers of the matched representative fingerprints, the unmatched fingerprints will be compared with the on-disk chunk fingerprint index. Finally, the chunks corresponding to the unmatched fingerprints are stored in an open unfilled container or a new container. Our similarity-index based optimization can achieve high deduplication throughput with less system RAM resource overhead by preserving strong chunk-fingerprint cache locality over container management.

4 Evaluation

We have implemented a prototype of Σ-Dedupe as a middleware in user space using C++ and pthreads, on the Linux platform. We evaluate the parallel deduplication efficiency in the single-node multi-core deduplication server with real system implementation, while use trace-driven simulation to demonstrate how Σ-Dedupe outperforms the state-of-the-art cluster deduplication techniques by achieving a high cluster-wide capacity saving that is very close to the extremely high-overhead stateful approach at a slightly higher overhead than the highly scalable stateless approach, while maintaining a scalable performance in large cluster deduplication. In addition, we conduct sensitivity studies to answer the following important design questions:

- What is the best chunk size for the single-node deduplication to achieve high deduplication efficiency?
- How does similarity index lock granularity affect the representative fingerprint index performance?
- How sensitive is the cluster deduplication ratio to handprint size?

4.1 Evaluation Platform and Workload

We use two commodity servers to perform our experiments to evaluate parallel deduplication efficiency in single-node deduplication servers. All of them run Ubuntu 11.10 and use a configuration with 4-core 8-thread Intel X3440 CPU running at 2.53 GHz and 16GB RAM and a SAMSUNG 250GB hard disk drive. One server serves as

both the backup client and director, and the other as the deduplication server. Our prototype deduplication system uses GBit Ethernet for internal communication. To achieve high throughput, our backup client component is based on an event-driven, pipelined design, which utilizes an asynchronous RPC implementation via message passing over TCP streams. All RPC requests are batched in order to minimize the round-trip overheads. We also perform simulation on one of the two servers to evaluate the cluster deduplication techniques.

Table 2. The workload characteristics of the real-world datasets and traces

Datasets	Size (GB)	Deduplication Ratio
Linux	160	8.23(CDC) / 7.96(SC)
VM	313	4.34(CDC) / 4.11(SC)
Mail	526	10.52(SC)
Web	43	1.9(SC)

We collect two kinds of real-world datasets and two types of application traces for our experiments. The Linux dataset is a collection of Linux kernel source code from versions 1.0 through 3.3.6, which is downloaded from the website [19]. The VM dataset consists of 2 consecutive monthly full backups of 8 virtual machine servers (3 for Windows and 5 for Linux). The mail and web datasets are two traces collected from the web-server and mail server of the CS department in FIU [20]. The key workload characteristics of these datasets are summarized in Table 2. Here, the "size" column represents the original dataset capacity, and "deduplication ratio" column indicates the ratio of logical to physical size after deduplication with 4KB fixed chunk size in static chunking (SC) or average 4KB variable chunk size in content defined chunking (CDC).

4.2 Evaluation Metrics

The following evaluation metrics are used in our evaluation to comprehensively assess the performance of our prototype implementation of Σ-Dedupe against the state-of-the-art cluster deduplication schemes.

Deduplication efficiency: A simple metric that encompasses both deduplication effectiveness and overhead in single-node deduplication. It is well understood that the deduplication efficiency is proportional to deduplication effectiveness that can be defined by deduplication ratio (*DR*), which is the ratio of logical size to physical size of the dataset, and inversely proportional to deduplication overhead that can be measured by deduplication throughput (*DT*), which is the ratio of logical dataset size to deduplication process time. Based on this understanding and to better quantify and compare deduplication efficiency of a wide variety of deduplication techniques, we adopt a metric, called "bytes saved per second", which is first defined in [13], to measure the efficiency of different deduplication schemes in the same platform by feeding a given dataset. It is calculated by the difference between the logical size *L*

and the physical size P of the dataset divided by the deduplication process time T. So, deduplication efficiency (DE) can be expressed in Eq. (6).

$$DE = \frac{L-P}{T} = (1-\frac{1}{DR}) \times DT \quad (6)$$

Normalized deduplication ratio: The metric is designed for cluster deduplication effectiveness. It is equal to the cluster deduplication ratio divided by deduplication ratio achieved by a single-node, exact deduplication system. This is an indication of how close the deduplication ratio achieved by a cluster deduplication method is to the ideal cluster deduplication ratio.

Normalized effective deduplication ratio: A single utility measure that considers both cluster-wide deduplication effectiveness and storage imbalance. It is equivalent to normalized deduplication ratio divided by the value of 1 plus the ratio of standard deviation σ of physical storage usage to average usage α in all deduplication servers, similar to the metric used in [6]. According to the definition of normalized deduplication ratio by cluster deduplication ratio (CDR) and single-node deduplication ratio (SDR), normalized effective deduplication ratio ($NEDR$) can be expressed in Eq. (7). It indicates how effective the data routing schemes are in eliminating the deduplication node information island.

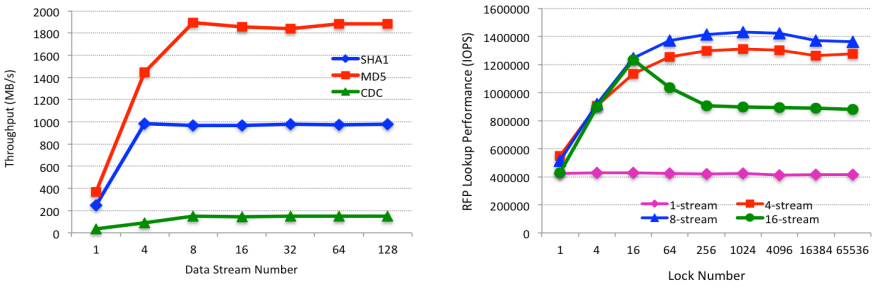
$$NEDR = \frac{CDR}{SDR} \times \frac{\alpha}{\alpha+\sigma} \quad (7)$$

Number of fingerprint index lookup messages: An important metric for system overhead in cluster deduplication, which significantly affects the cluster system scalability. It includes inter-node messages and intra-node messages for chunk fingerprint lookup, both of which can be easily obtained in our simulation to estimate cluster deduplication overhead.

4.3 Parallel Deduplication Efficiency on Single-Node Server

As deduplication is a resource intensive task, we develop parallel deduplication on multiple data streams for each node with multi-thread programming in pthreads to leverage the compute capabilities of multi-core or many-core processor of modern commodity servers. In our design, we adopt the RAM file system to store the workload and avoid unique data write to disks to eliminate the disk I/O performance bottleneck, due to our low disk I/O configuration. Meanwhile, we assign a deduplication thread for each data stream to read in parallel different files that are stored in RAM to create multiple data streams. We measure the throughput of parallel chunking and hash fingerprinting at backup clients as a function of the number of data streams. Considering the fact that static chunking has negligible overhead, we only test Rabin hash based content defined chunking (CDC) for chunking throughput by implementing it based on the open source code in Cumulus [21] with 4KB average chunk size. The implementation of the hash fingerprinting is based the OpenSSL library. Figure 4(a) shows the experiment results of CDC and MD5/SHA-1 based fingerprinting in the backup client. The throughput of chunking and fingerprinting can scale almost linearly and reach their peak values (148MB/s for CDC, 980MB/s for SHA-1 finger-

printing and 1890MB/s for MD5 fingerprinting) with 4 or 8 data streams, since the processor at the client is a 4-core 8-thread CPU. To find more data redundancy in backup datasets, CDC may affect the performance of deduplication for its low throughput and high deduplication time. We select SHA-1 to reduce the probability of hash collision even though its throughput is only about a half that of MD5.



(a) Chunking and fingerprinting throughput in backup client

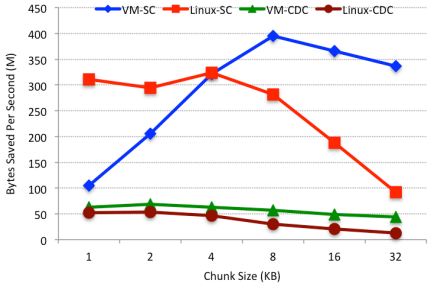
(b) The performance of similarity index parallel lookup

Fig. 4. Intra-node parallel deduplication performance

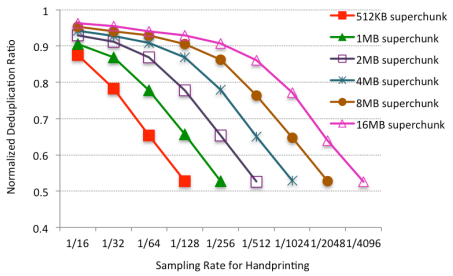
To exploit the multi-core or many-core resource of the deduplication server node, we also develop parallel similarity index lookup in individual deduplication servers. For our multiple-data-stream based parallel deduplication, each data stream has a deduplication thread, but all data streams share a common hash-table based similarity index in each deduplicaiton server. We lock the hash-table based similarity index by evenly partitioning the index at the single-hash-bucket or multiple-contiguous-hash-bucket granularity to support concurrent lookup. We test the performance of the parallel similarity index when all index data is loaded in memory. To avoid the possible performance bottleneck of a single backup client, we feed the deduplication server with chunk fingerprints generated in advance. Figure 4(b) shows the performance of the parallel index lookup for multiple data streams as a function of the number of locks. When the number of locks is greater than 1024, the performance drops as the lock overhead becomes non-negligible. 8 data steams still perform the best because the CPU supports 8 concurrent threads, while the performance of 16 streams drops when the number of locks is larger than 16 because of the overhead of thread context switching that causes data swapping between cache and memory.

According to the metric defined in section 4.2, we measure the deduplication efficiency in a configuration with a single backup client and a single deduplication server to show the tradeoff between deduplication effectiveness and overhead. To eliminate the impact of the disk bottleneck, we store the entire workload in memory and perform the deduplication process to skip the unique-data-chunk store step. The deduplication effectiveness is affected by the data chunking method and the selected chunk size. High deduplication effectiveness can be achieved by using small chunk size and the CDC scheme instead of the Static Chunking (SC) scheme. This high effectiveness, however, comes at the expense of the increased amount of metadata necessary to manage the increased number of chunks and variable chunk size, negatively impact-

ing system's performance. To assess the impact of the metadata overhead on deduplication efficiency, we measure "Bytes Saved Per Second", the deduplication efficiency as defined in Section 4.2, as a function of the chunk size. The results in Figure 5(a) show that SC outperforms CDC in deduplication efficiency due to the former's low overhead in data chunking. The deduplication efficiency is dynamically changing with the selected chunk size, and also depends on the workload. The single deduplication server can achieve the highest deduplication efficiency when the chunk size is 4KB for statically chunked Linux workload, 8KB for statically chunked VM workload and 2KB for both workloads with CDC. As a result, we choose to perform chunking with the SC scheme and select 4KB as the chunk size in the following experiments for high deduplication efficiency.



(a) Deduplication efficiency in single server



(b) Deduplication effectiveness as a function of the sampling rate and super-chunk size

Fig. 5. Sensitivity study on chunk size and handprint size

In our Σ-Dedupe design, handprinting plays a key role in similarity-index based deduplication optimization since the container ID is indexed by the representative fingerprints of a handprint. Handprinting is a novel use of deterministic sampling, where we define the *handprint-sampling rate* as the ratio of handprint size to the total number of chunk fingerprints in a super-chunk. This sampling rate affects both the deduplication effectiveness and RAM usage in each node. We turn off the traditional chunk index lookup module in our prototype, and conduct a series of experiments to demonstrate the effectiveness of the handprint-based local deduplication in Σ-Dedupe. Figure 5(b) shows the deduplication ratio produced by applying our similarity-index-only optimization (without traditional chunk index) to the Linux workload, normalized to that of the traditional single-node exact deduplication with a chunk size of 4KB, as a function of the handprint-sampling rate and super-chunk size. As can be seen, the deduplication ratio falls off as the sampling rate decreases and as the super-chunk size decreases, and the "knee" point for the 16MB super-chunk at the sample rate of 1/512 is a potentially best tradeoff to balance deduplication effectiveness and RAM usage, and it translates to a handprint size of 8. Meanwhile, the results further suggest that, interestingly, the deduplication ratio remains roughly constant if the sampling rate is halved and super-chunk size is doubled at the same time. As a result, we can find that 8 representative fingerprints in a handprint are sufficient to achieve a deduplication ratio that is close to that of the exact deduplication approach with high

RAM utility. Furthermore, and importantly, Σ-Dedupe only uses 1/32 of the RAM capacity required by the traditional chunk index to store our similarity index to achieve about 90% of the deduplication effectiveness when the super-chunk and handprint sizes are 1MB and 8 respectively. A first-order estimate of RAM usage, based on our earlier analysis, indicates that, comparing with the intra-node deduplicaition scheme of the EMC super-chunk based data routing—DDFS [3], and the intra-node deduplication of Extreme Binning, for a 100TB unique dataset with 64KB average file size, and assuming 4KB chunk size and 40B index entry size, DDFS requires 50GB RAM for Bloom filter, Extreme Binning uses 62.5GB RAM for file index, while our scheme only needs 32GB RAM to maintain similarity index. We can further reduce the RAM usage by adjusting super-chunk size or handprint size with the corresponding deduplication effectiveness loss as indicated in Figure 5(b).

4.4 Cluster-Deduplication Efficiency

We route data at the super-chunk granularity to preserve data locality for high performance of cluster-wide deduplication, while performing deduplication at the chunk granularity to achieve high deduplication ratio in each server locally. Since the size of the super-chunk is very sensitive to the tradeoff between the index lookup performance and the cluster deduplication effectiveness, as demonstrated by the sensitivity analysis on super-chunk size in [6], we choose the super-chunk size of 1MB to reasonably balance the conflicting objectives of cluster-wide system performance and capacity saving. In this section, we first conduct a sensitivity study to select an appropriate handprint size for our Σ-Dedupe scheme, and then compare our scheme with the state-of-the-art approaches that are most relevant to Σ-Dedupe, including EMC's super-chunk based data Stateful and Stateless routing and Extreme Binning, in terms of the effective deduplication ratio, normalized to that of the traditional single-node exact deduplication, and overhead measured in number of fingerprint index lookup messages. We emulate each node by a series of independent fingerprint lookup data structures, and all results are generated by trace-driven simulations on the four datasets under study.

Handprint-based Stateful routing can accurately direct similar data to the same deduplication server by exploiting data similarity. We conduct a series of experiments to demonstrate the effectiveness of cluster deduplication by our handprint-based deduplication technique with the super-chunk size of 1MB on the Linux workload. Figure 6 shows the deduplication ratio, normalized to that of the single-node exact deduplication, as a function of the handprint size. As a result, Σ-Dedupe becomes an approximate deduplication scheme whose deduplication effectiveness nevertheless improves with the handprint size because of the increased ability to detect resemblance in super-chunks with a larger handprint size (recall Section 2.2). We can see that there is a significant improvement in normalized deduplication ratio for all cluster sizes when handprint size is larger than 8. This means that, for a large percentage of super-chunk queries, we are able to find the super-chunk that has the largest content overlap with the given super-chunk to be routed by our handprint-based routing scheme. To strike a sensible balance between the cluster-wide deduplication ratio and

system overhead, and match the handprint size choice in single-node, we choose a handprint consisting of 8 representative fingerprints in the following experiments to direct data routing on super-chunks of 1MB in size.

To compare Σ-Dedupe with the existing data routing schemes in cluster deduplication, we use the effective deduplication ratio (EDR), normalized to the deduplication ratio of the single-node exact deduplication, to evaluate the cluster-wide deduplication effectiveness with the load-balance consideration. We compare our Σ-Dedupe scheme with the state-of-the-art cluster-deduplication data routing schemes of Extreme Binning (ExtremeBin), EMC's stateless (Stateless) and EMC's stateful (Stateful) routing schemes, across a range of datasets. Figure 8 plots EDR as a function of the cluster size for the four datasets and four algorithms. Because the last two traces, Mail and Web, do not contain file-level information, we are not able to perform the file-level based Extreme Binning scheme on them. In general, Σ-Dedupe can achieve a high effective deduplication ratio very close that achieved by the very costly Stateful data routing. More specifically, the Σ-Dedupe scheme achieves 90.5%~94.5% of the EDR obtained by the Stateful scheme for a cluster of 128 server nodes on the four datasets, while this performance margin narrows to 96.1%~97.9% when averaging over all cluster sizes, from 1 through 128. Stateless routing consistently performs worse than Σ-Dedupe and Stateful routing due to its low cluster-wide data reduction ratio and unbalanced capacity distribution. Extreme Binning underperforms Stateless routing on the VM dataset because of the large file size and skewed file size distribution in the VM dataset, workload properties that tend to render Extreme Binning's similarity detection ineffective. Σ-Dedupe outperforms Extreme Binning in EDR by up to 32.8% and 228.2% on the Linux and VM datasets respectively for a cluster of 128 nodes. For the four datasets, Σ-Dedupe is better than Stateless routing in EDR by up to 25.6%~271.8% for a cluster of 128 nodes. As can be seen from the trend of curves, these improvements will likely be more pronounced with cluster sizes larger than 128.

In cluster deduplication systems, fingerprint lookup tends to be a persistent bottleneck in each deduplication server because of the costly on-disk lookup I/Os, which often adversely impacts the system scalability due to the consequent high communication overhead from fingerprint lookup. To quantify this system overhead, we adopt a metric used in [6], the number of fingerprint-lookup messages. We measure this metric by totaling the number of chunk fingerprint-lookup messages on the two real datasets of Linux and VM, for the four cluster deduplication schemes. As shown in Figure 7 that plots the total number of fingerprint-lookup messages as a function of the cluster size, Σ-Dedupe, Extreme Binning and Stateless routing have very low system overhead due to their constant fingerprint-lookup message count in the cluster deduplication process, while the number of fingerprint-lookup messages of Stateful routing grows linearly with the cluster size. This is because Extreme Binning and Stateless routing only have 1-to-1 client-and-server fingerprint-lookup communication for source deduplication due to their stateless designs. Stateful routing, on the other hand, must send the fingerprint lookup requests to all nodes, resulting in 1-to-all communication that causes the system overhead to grow linearly with the cluster size even though it can reduce the overhead in each node by using a sampling scheme. As de-

scribed in Algorithm 1, the main reason for the low system overhead in Σ-Dedupe is that the pre-routing fingerprint-lookup requests for each super-chunk only need to be sent to at most 8 candidate nodes, and only for the lookup of representative fingerprints, which is 1/32 of the number of chunk fingerprints, in these candidate nodes. The total number of fingerprint-lookup messages for Σ-Dedupe is the sum of after-routing message number, which is almost the same as Extreme Binning and Stateless routing, and pre-routing message number, which is a quarter (8×1/32) that of after-routing. So the fingerprint lookup message overhead will not exceed 1.25 times that of Stateless routing and Extreme Binning in all cluster sizes.

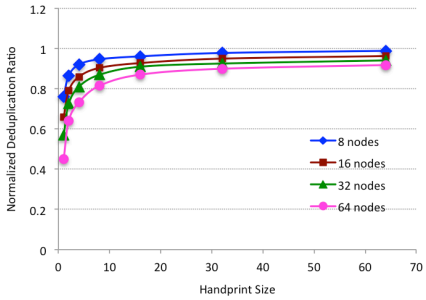


Fig. 6. Cluster deduplication ratio (DR), normalized to that of single-node exact deduplication, as a function of handprint size

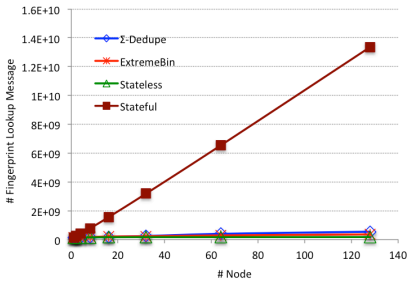


Fig. 7. System overhead in terms of the number of fingerprint-lookup messages

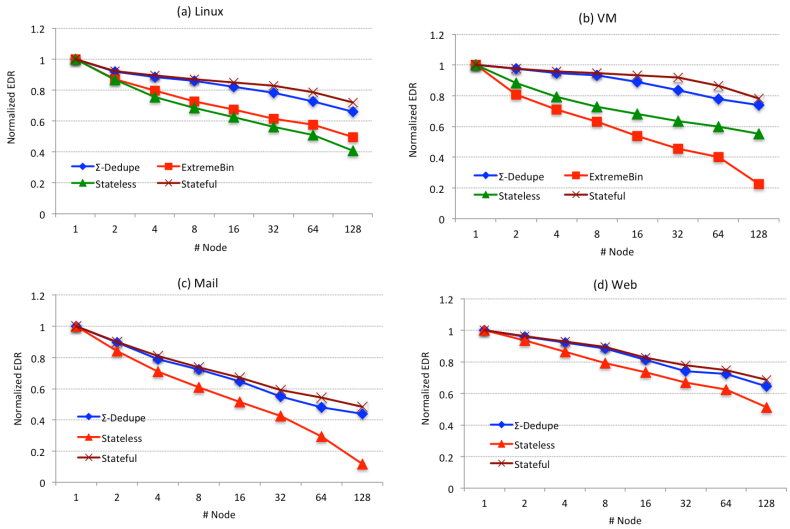


Fig. 8. Effective deduplication ratio (EDR), normalized to that of single-node exact deduplication, as a function of cluster size on four workloads

5 Conclusion

In this paper, we describe Σ-Dedupe, a scalable inline cluster deduplication framework for Big Data protection, which achieves a tradeoff between scalable performance and cluster-wide deduplication effectiveness by exploiting data similarity and locality in backup data streams. It adopts a handprint-based local stateful routing algorithm to route data at the super-chunk granularity to reduce cross-node data redundancy with low overhead, employs similarity index based optimization to improve deduplication efficiency in each node with very low RAM usage. Our real-world dataset-driven evaluation clearly demonstrates Σ-Dedupe's significant advantages over the state-of-the-art cluster deduplication schemes for large clusters in the following important two ways. First, it nearly (over 90%) achieves the cluster-wide deduplication ratio of the extremely costly and poorly scalable Stateful cluster deduplication scheme but only at a slightly higher overhead than the highly scalable Stateless and Extreme Binning schemes. Second, it significantly improves the Stateless and Extreme Binning schemes in the cluster-wide effective deduplication ratio while retaining the latter's high system scalability for low overhead. Meanwhile, high parallel deduplication efficiency can be achieved in each node by exploiting similarity and locality in backup data streams.

References

1. Villars, R.L., Olofson, C.W., Eastwood, M.: Big Data: What It Is and Why You Should Care. White Paper, IDC (2011)
2. Kolodg, C.J.: Effective Data Leak Prevention Programs: Start by Protecting Data at the Source-Your Databases. White Paper, IDC (2011)
3. Zhu, B., Li, K., Patterson, H.: Avoiding the Disk Bottleneck in the Data Domain Deduplication File System. In: Proc. of USENIX FAST (2008)
4. Gantz, J., Reinsel, D.: The Digital Universe Decade-Are You Ready? White Paper, IDC (2010)
5. Biggar H.: Experiencing Data De-Duplication: Improving Efficiency and Reducing Capacity Requirements. White Paper. The Enterprise Strategy Group (2007)
6. Dong, W., Douglis, F., Li, K., Patterson, H., Reddy, S., Shilane, P.: Tradeoffs in Scalable Data Routing for Deduplication Clusters. In: Proc. of USENIX FAST (2011)
7. Douglis, F., Bhardwaj, D., Qian, H., Shilane, P.: Content-aware Load Balancing for Distributed Backup. In: Proc. of USENIX LISA (2011)
8. Bhagwat, D., Eshghi, K., Long, D.D., Lillibridge, M.: Extreme Binning: Scalable, Parallel Deduplication for Chunk-based File Backup. In: Proc. of IEEE MASCOTS (2009)
9. Dubnicki, C., Gryz, L., Heldt, L., Kaczmarczyk, M., Kilian, W., Strzelczak, P., Szczepkowski, J., Ungureanu, C., Welnicki, M.: HYDRAstor: a Scalable Secondary Storage. In: Proc. of USENIX FAST (2009)
10. Bhagwat, D., Eshghi, K., Mehra, P.: Content-based Document Routing and Index Partitioning for Scalable Similarity-based Searches in a Large Corpus. In: Proc. of ACM SIGKDD (2007)
11. Yang, T., Jiang, H., Feng, D., Niu, Z., Zhou, K., Wan, Y.: DEBAR: a Scalable High-Performance Deduplication Storage System for Backup and Archiving. In: Proc. of IEEE IPDPS (2010)

12. Kaiser, H., Meister, D., Brinkmann, A., Effert, S.: Design of an Exact Data Deduplication Cluster. In: Proc. of IEEE MSST (2012)
13. Fu, Y., Jiang, H., Xiao, N., Tian, L., Liu, F.: AA-Dedupe: An Application-Aware Source Deduplication Approach for Cloud Backup Services in the Personal Computing Environment. In: Proc. of IEEE Cluster (2011)
14. Jaccard Index. http://en.wikipedia.org/wiki/Jaccard_index.
15. Broder, A.Z., Charikar, M., Frieze, A.M., Mitzenmacher, M.: Min-wise Independent Permutations. In: Journal of Computer and System Sciences, 60(3): 630–659, Elsevier, Amsterdam (2000)
16. Eshghi, K., Tang, H.K.: A framework for Analyzing and Improving Content-based Chunking Algorithms. Technical Report, Hewlett Packard (2005)
17. Wallace, G., Douglis, F., Qian, H., Shilane, P., Smaldone, S., Chamness, M., Hsu, W.: Characteristics of Backup Workloads in Production Systems. In: Proc. of FAST (2012)
18. Xia, W., Jiang, H., Feng, D., Hua Y.: Silo: a Similarity-locality based Near-exact Deduplication Scheme with Low RAM Overhead and High Throughput. In: Proc. of USENIX ATC (2011)
19. The Linux Kernel Archives. http://www.kernel.org/
20. FIU IODedup Traces. http://iotta.snia.org/traces/391
21. Vrable, M., Savage, S., Voelker, G.M.: Cumulus: Filesystem Backup to the Cloud. In: Proc. of USENIX FAST (2009)
22. Efstathopoulos, P.: File Routing Middleware for Cloud Deduplication. In: Proc. of ACM CloudCP (2012)
23. IBM ProtecTIER Deduplication Gateway. http://www-03.ibm.com/systems/storage/tape/ts7650g/index.html
24. EMC Data Domain Global Deduplication Array. http://www.datadomain.com/products/global-deduplication-array.html
25. SEPATON S2100-ES2. http://www.sepaton.com/products/SEPATON_ES2.html