Rangoli: Space management in deduplication environments

P. C. Nagesh
NetApp Inc
nageshc@netapp.com

Atish Kathpal
NetApp Inc
atish.kathpal@netapp.com

ABSTRACT

Space management is the activity of monitoring and ensuring adequate free space on all volumes in a clustered storage system. Volumes that exceed used space limits are typically relieved by migrating a part of their data to other under utilized volumes. Without deduplication, space reclamation is simple as one has to just migrate as much data as the desired space reclamation. However, in deduped volumes there is no direct relation between the logical size of the file and the physical space occupied by it. Therefore, optimal space reclamation is hard as: a)migrating few files may produce little or zero bytes of free space, but still incur significant network costs. b)migrating a heavily shared file destroys the disk sharing relationships in that volume and increases the physical space consumption of that dataset.

In this work, we have designed and built a fast and efficient tool Rangoli, that identifies the optimal set of files for space reclamation in a deduped environment. It can scale to millions of files and terabytes of data, running in tens of minutes. We show by experimenting on real world datasets, that alternate strategies such as those based on finding unique files or using *MinHash*, impact physical space consumption by a wide margin (up to 35 times) as compared to Rangoli.

Categories and Subject Descriptors

D.4.3 [**Operating Systems**]: File Systems Management; D.4.2 [**Operating Systems**]: Storage Management

General Terms

space management, capacity balancing

1. INTRODUCTION

Space management is a critical component of storage system administration. Alerts (such as *E_NO_SPACE*) are issued whenever the free space in a volume drops below certain limits. Often, the only solution is to migrate away a part of the data from that volume. Simpler options such as adding disks are not always feasible as hard limits exist on the capacity of a volume for performance reasons. Further, this requires manual intervention which may not be timely. Therefore, many multi-node clustered storage systems such as CEPH [12] and Panasas [13] incorporate automated mechanisms that monitor the free space levels in its constituent volumes and transparently migrate a part of an over utilized volume to a sparsely populated volume. We refer to this task of reducing physical space consumption of a volume as *space reclamation.* Space reclamation in non deduped environments is simpler. Here, adding or deleting a file from a volume leads to guaranteed changes in the used space of the volume by an amount equal to the logical size of the file.

Deduplication [10, 9, 2] is a widely adopted storage efficiency technique that introduces disk sharing between the files in a volume. The effect of file addition or deletion, on the used space of a deduplicated volume is hard to predict as it depends on the extent of duplicate data of that file with the remainder of the volume. A seemingly full volume might still be able to accommodate a large file. Conversely, deletion of a large file might free up little or no space. Migrating such a file leads to wasteful network costs without achieving the desired space reclamation.

Our primary insight is that similar files need to be migrated together. We observed that in a highly deduped volume, often migrating a file produced minimal free space, corresponding to its meta data size only and the data block were not freed as they were being shared with other files. Also, we noticed that administrators typically preferred to migrate unique files, as these guaranteed much greater space reclamation. This is because there are many data blocks that are only owned by the unique file and therefore readily freed. This led us to devise a strategy of finding groups of files that share little or no data with the remainder of the volume. Such groups in effect act as a large unique file with reference to the remainder of the volume and form optimal candidates for space reclamation. Our contributions can be summarized as:

- A novel solution for space reclamation in deduped environments that is fast and scalable and tested on real world datasets.
- A deterministic solution that reports the exact metrics before the actual migration (i.e we find the exact space reclamation they produce and the associated penalties, such as network costs and physical space consumption).
- We find optimal datasets for space reclamation that

SYSTOR '13, June 30 - July 02 2013, Haifa, Israel
Copyright 2013 ACM 978-1-4503-2116-7/13/06 ...$15.00.

are significantly and consistently better than alternatives namely the strategy of migrating unique files and the strategy based on MinHash [5].

2. BACKGROUND AND RELATED WORK

Our clustered storage incorporates specialized meta data management (similar to CEPH [12]) that hides the actual location of the file from the end user, allowing for transparent movement of files across volumes. Rangoli identifies optimal groups of files for space reclamation and reports their exact metrics to a higher subsystem that decides the final migration plan. The actual migration can take hours but the selection of files for migration is expected to complete within tens of minutes. We leverage the fingerprint database (FPDB) to compute the extent of disk sharing across files in a volume. The alternative is to crawl the filesystem metadata which is very time consuming. FPDB is essentially a mapping between data blocks and their cryptographic strength fingerprints maintained by many deduplication engines [1, 8]. In rare cases, common fingerprints do not imply common data blocks. This can happen due to hash collision, a very low probable event that is ignored.

In migrating data, we have the following options with reference to storage efficiency:

1. Source centric: Select groups of files at source that have a high degree of disk sharing. Migrate them together to carry forward the disk sharing opportunities between them in their new destination.
2. Destination aware: Pick files at the source that potentially have maximum duplicate data at some destination volume and route it accordingly.

Option(1) is primarily about *storage efficiency preservation* and option(2) is an aggressive strategy that seeks to further increase the global storage efficiency. The following factors affect the choice between the two strategies. Often a space starved volume is an indicator of growing data set size and calls for introduction of newer and empty volumes. Here, the destination is a fixed empty volume and a source centric strategy is the only option. When the destination has to be chosen from older volumes, we need per-volume content indicators such as bloom filters [14] to choose the best destination. This could be achieved by computing bloom filters on the fly, or by periodic maintenance and updation for every data ingest. The costs of this approach requires careful consideration against the gains which is subject to the distribution of data in the cluster.

Space reclamation is a critical but a less frequent problem where the primary requirement is to quickly relieve an ailing volume. In this work, we focussed on a source centric and destination agnostic solution which accomplishes space reclamation with maximum storage efficiency preservation.

2.1 Related Work

Deduplication is a well researched area [14, 10, 6, 7] but its impact on space management issues are not well studied. Storage rebalancing by migrating virtual bins is a closely related work from Data Domain [4]. They group super chunks to virtual bins using hashing functions such as MinimumHash. The super chunks are collections of similar stripes of data drawn from backup streams. However, file striping is not supported in our environment and whole files are to be migrated. A similar application of hashing techniques is for similarity searches and document routing to a set of partitions [3]. Min wise independent permutations (MinHash) is used over document features to ensures that similar documents overlap in their set of partitions. In contrast, we are interested in assigning files to a disjoint set of partitions in our work. Also, the notion of similarity is drawn from disk sharing relationships across files and not the semantic content of the documents. We show in our evaluation that grouping files using hashing techniques such as MinimumHash [4] and approximate MinHash [5] does not work well for space reclamation.

3. SOLUTION OVERVIEW AND DETAILS

3.1 Overview

Our solution is motivated by the observation that migrating similar files is better for preserving storage efficiency. If three files F1, F2 and F3 (Figure 1) share a lot of data, migrating them together is essential to free up the common blocks held by them. Further, they will continue to share data in their new destination thus preserving storage efficiency.

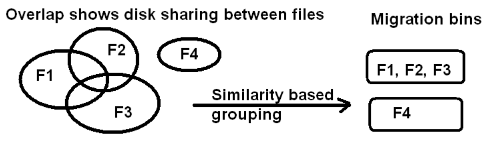
Overlap shows disk sharing between files
Migration bins
F1
F2
F3
F4
Similarity based grouping
F1, F2, F3
F4

Figure 1: Dividing the dataset into migration bins

We seek to partition the dataset such that most of the data sharing is between files within the same partition and files across partitions have little or no data sharing. These partitions are referred to as *migration bins* and are the primary units for space reclamation.The partitioning seeks to produce migration bins that offer the desired space reclamation with least impact on storage efficiency and network costs. These are measured using the following metrics :

3.1.1 Metric definitions and notations

1. Space Reclamation (SR): SR is the difference in the total used (physical) space of the source volume before and after migration.
2. Cost of Migration (CM): CM is the number of blocks transmitted over the network. This is the number of unique blocks in the migration bin.
3. Migration Utility (MU): MU is defined as $\frac{SR}{CM}$. Ideally, we desire high MU as it signifies the amount of accomplished objective (SR) for the cost incurred (CM).
4. Physical space bloat (PSB): PSB is the ratio of increase in the physical space consumption of the dataset to its original space consumption. When all disk sharing relationships are preserved, no increase in physical space consumption occurs. Therefore, we seek low values of PSB.
   Note that these metrics are source centric measurements, not accounting for any duplication across the source and destination volumes.

3.2 Algorithms

Our solution consists of the following three steps:

1. FPDB processing: We process the FPDB to compute the extent of data sharing across files and then represent this information as a bipartite graph.
2. Migration binning: Graph partitioning is then performed to obtain K migration bins when the desired space reclamation is $\frac{1}{K}$ of the volume space. The partitioning is such that each migration bin is approximately equal in size.
3. Qualification of migration bins: Finally we compute the metrics for each migration bin and the best amongst them is then chosen for space reclamation.

We describe these steps in detail in the following sections.

3.2.1 FPDB processing

Algorithm 1 finds the amount of disk sharing between files. Common blocks between files are given by common fingerprints in the FPDB and the corresponding block lengths are summed up to compute the disk sharing. The output is a bipartite graph with the left side nodes representing the files (or inodes). The algorithm combines multiple fingerprints or data blocks with the same set of owner inodes into a single right side node of the bipartite graph. The weight of the node denotes the amount of disk sharing.

```
Input: FPDB file (sorted by fingerprint order)
Output: Bipartite graph
Initialize RightNodeWeights=$\emptyset$, Neighbors = $\emptyset$,
Nodelist = $\emptyset$, currentFingerprint = 0,
previousFingerprint = 0
foreach line in FPDB of the form <fingerprint, block
len, inode> do
    previousFingerprint := currentFingerprint ;
    currentFingerprint := fingerprint ;
    if previousFingerprint = currentFingerprint then
        Nodelist := Nodelist $\cup$ inode;
    end
    else
        create a unique identifier for the set of
        nodes in Nodelist;
        nodeId := hash(nodelist) ;
        RightNodeWeights [ nodeId] :=
        RightNodeWeights [ nodeId] + block len ;
        Neighbors [ nodeId ] := nodelist ;
        Nodelist := $\emptyset$ ;
    end
end
```

Algorithm 1: FPDB processing to produce a graphical representation of the dataset.

3.2.2 Migration binning

The graphical modeling obtained from the previous step, is a loss less depiction of the exact data sharing extents across groups of files. Using this graph, we partition the set of files into disjoint subsets. These subsets are the final migration bins.

The algorithm starts with every file in its own partition and progressively merging two or more partitions. Further, partitions that are chosen for merger are such that there exists little or no data sharing across them. This is achieved by traversing the list of data sharing extents (right-side nodes on the bipartite graph), in the descending order of their weights. The detailed steps are as follows:

1. Input: Bipartite graph, K - no of bins , Output: K migration bins
2. Initialize every file to be in its own partition
3. for every node d on the right side of the bipartite graph in the decreasing order of their weights:
   (a) follow the edges and pick the set of neighboring nodes on the left side of the bipartite graph. This set is denoted as : $N \leftarrow \{x|\exists\text{edge between d and x}\}$
   (b) Find P the minimal set of partitions corresponding to N. i.e $P \leftarrow \{p|\exists m \in N \text{ and m is in partition p}\}$
   (c) List the partitions in P in the non-decreasing order of their sizes. If there are two or more partitions that are consecutive in the above listing and whose combined size is $<= \frac{1}{K}$ of the (logical) size of the dataset, merge them. Repeat this step until no more merging is possible.
4. Assign all files in a partition to a single migration bin. If there are more than K partitions, combine the smaller ones to get K migration bins.

3.2.3 Qualifying the migration bins

In this final step of our solution, we compute the exact metrics of the migration bins found in previous step, as described below:

1. Input: Bipartite graph, Migration bins (P)
2. Output: Metrics for each migration bin
3. Initialize the following members for each bin p in P:
   (a) Logical size(p) = sum total of the logical sizes of each file belonging to bin p
   (b) InternalSharings(p) = 0 . This denotes the extent of data sharing within a bin
   (c) SharingAcross(p) = 0 . This denotes the extent of data sharing of the bin p with the remainder of the dataset.
4. for each right side node r in the bipartite graph, do :
   (a) follow the edges and pick the set of neighboring nodes on the left side of the bipartite graph. This set is denoted as : $N \leftarrow \{x|\exists\text{edge between r and x}\}$
   (b) P is the minimal set of partitions corresponding to N. i.e $P \leftarrow \{p|\exists m \in N \text{ and m is in partition p}\}$
   (c) for each bin p in P , do :
      i. InternalSharings(p)+=(x-1)*weight(r), where x is the number of nodes in N belonging to bin p
      ii. if the size of set P is > 1 then do SharingAcross(p) += weight(r)
5. Output the metrics for each bin p :
   (a) CM (p) = Logical size(p) - Internal sharing (p)
   (b) SR (p) = CM(p) - SharingAcross (p)
   (c) MU (p) = $\frac{SR(p)}{CM(p)}$
   (d) PSB (p) = $\frac{CM(p)-SR(p)}{S}$ where S is the physical size occupied by the dataset before migration

The algorithm works on the bipartite graph by computing the amount of disk sharing within and across migration bins. InternalSharings is the amount of disk sharing within a migration bin. Upon migration these blocks are completely freed. As only unique blocks in a bin are transferred, we can compute CM as the difference in the logical size of the migration bin and its InternalSharings count. The counter SharingAcross represents the amount of disk sharing across migration bins. Upon migration, they are not freed at the source volume, but produce a copy on the destination volume. Therefore, SR is the difference in the number of blocks moved (CM) and those that were not freed (SharingAcross). Further, this quantity represents the increase in the physical size occupied by the dataset after migration and is used to measure PSB. Of the K migration bins obtained, we select those bins whose space reclaimation is close to the desired value (i.e within a tolerance of 10%). Amongst these, the bin with the best metrics (PSB is preferred over MU) is chosen for the final migration.

3.3 Time and space complexity analysis

The deduplication engine [1] maintains a sorted FPDB that is accessible in the administrator mode. The FPDB contains one fingerprint record for every logical block of the file. The number of fingerprints ($f = \frac{dataset\,size}{average\,block\,size}$) is proportional to the logical size of the dataset. The total number of files in the dataset is denoted by n. These form the left side nodes of the bipartite graph. The total number of disk blocks is lesser than f owing to deduplication and represents the physical space occupied by the dataset. Multiple disk blocks that share the same set of owner files are jointly represented by a single right side node, that makes the bipartite graph small enough to fit in memory. The number of such right side nodes is denoted by m. Edges are drawn from such right side nodes to connect to the owner files, represented as the left side nodes of the graph. The number of such edges is denoted by e.

In FPDB processing we make a single scan over the FPDB file and its time complexity is linear in f. Its final output is the bipartite graph with m right side nodes that are listed in the decreasing order of their weights. This sorting step incurs an additional cost of $O(mlgm)$. Its space complexity is the size of the bipartite graph that it constructs in memory and is therefore linear in e with an adjacency list representation. The next steps of migration binning and qualification of migration bins also have a space complexity of $O(e)$, to hold the bipartite graph in memory.

In migration binning, we scan through each right side node and then follow up on the edges to reach the left side nodes. For each left side node we check which partition it belongs to. The partition information for the left side nodes are maintained using a union find datastructure. Therefore we do a maximum of e "find" operations. The number of partitions are initially n and reduce to K with a maximum of $n - K$ "union" operations. Using the weighted quick union find with path compression datastructure [11], we arrive at a time complexity of $O((n - K + e)lg^*n)$. Practically this is linear in e as $e > n$ and lg^*n is practically a constant.

In the next step of qualification of migration bins, we do a similar scan on the bipartite graph as in migration binning. However, there are no union-find operations. Every left side node is uniquely mapped to a migration bin and we just update the counters associated with each migration bin. Therefore, the time complexity for this step is $O(e)$.

Further, the first step of FPDB processing can be carried out on multiple parts of the FPDB independently by parallel threads and the resultant components of the bipartite graph can be easily combined. We found that the total running time reduced further in parallel mode (Section 4) .

4. EVALUATION

In this section we describe the primary evaluation objectives of our solution and compare Rangoli against alternative strategies. We describe the experimental set up and datasets used to measure these.

Algorithm	Time complexity	Space complexity
FPDB Processing	$O(f + mlgm)$	$O(e)$
Migration binning	$O(e)$	$O(e)$
Qualification of migration bins	$O(e)$	$O(e)$

Table 1: Time and space complexity analysis

4.1 Alternate strategies for comparison

A naive strategy (*Naive-du*) is to repeatedly pick the file with highest unique content (given by *du* in unix) and migrate enough such files to meet the desired space reclamation. The heuristic here is that selecting unique files reduces effects of disk sharing. The other insight that Rangoli is based on is that similar files need to be migrated together (Section 3). Similarity hashing based techniques are well studied for object placement [4]. Adapting these , we find file level hashes ($hash(f)$) and divide the dataset into K logical partitions based on these hashes (file f goes to partition $hash(f)\%K$). The best partition is picked to be a migration bin, similar to Rangoli's strategy. We evaluate against two types of similarity indicative hashes *Minimum Hash* [4] and *Min-Hash* [5].

4.2 Datasets

We evaluate and compare Rangoli using four real world datasets that are reflective of common deployments such as object stores, home directories, virtual desktop images, etc. The *Debian* dataset is a set of Debian ISO images, obtained by downloading weekly releases over a two month time frame. *HomeDir* is a set of home directory contents of several of our colleagues collected over a year. The *VMDK* dataset is a collection of virtual machine images used by our engineers over the past few years. *EngWebBurt* is a mixed dataset consisting of a combination of over a million engineering documents and bug reports from a live internal repository. To test the scalability of our algorithms further, we created two large synthetic datasets by replicating the files in the EngWebBurt dataset. Datasets *Synthetic1* and *Synthetic2* were obtained by replicating each file of EngWebBurt twice and thrice respectively. The properties of these datasets are described in Table 2.

4.3 Experimental Setup

We prototyped Rangoli as an application outside the storage controller. The datasets were hosted on a log structured storage controller with fixed block size deduplication ($4KB$) and the FPDB was exported through an administrator command.Rangoli and alternate strategies were implemented as Python applications processing the FPDB on a server class Linux machine with 2.5Ghz processor and 1GB of RAM.

4.4 Evaluation Objectives

The primary goal of space reclamation is to increase free space on the source volume by a desired amount. This has to be achieved with these additional objectives:

1. Flexibility: Any desired space reclamation objective should be achievable.
2. Impact on space consumption: Since no new data is added and the same dataset is being spread across multiple volumes, its physical space consumption should not increase. We seek low values of PSB (Section 3.1.1)

Dataset	Size	Dedupe	No. of files(n)	f	FPDB processing with 1 and 4 threads		m	e	Migration binning	Bin qualification	Total time with 1 and 4 threads	
HomeDir	74.7GB	49%	78K	19.5M	1min	18sec	28K	142K	5.4sec	0.8sec	1min	24sec
Debian	261GB	60%	447	68.4M	5min	1.3min	24K	116K	0.7sec	0.3sec	5min	1.3min
VMDK	2.4TB	62%	2.4K	644M	45min	13min	67K	664K	3.6sec	1.3sec	45min	13min
EngWebBurt	1.34TB	51%	4M	359M	26min	7min	391K	2.9M	1.5min	30sec	28min	9min
Synthetic1	2.68TB	77.5%	8M	719M	53min	14min	3.6M	12.5M	4min	2min	59min	20min
Synthetic2	4.02TB	85%	12M	1079M	77min	20min	3.6M	18M	5.4min	3min	85min	28min

Table 2: A detailed view of datasets and their running times (with the following notations: K for thousands, M for millions, min for minutes and sec for seconds)

to minimize adverse increases in physical space consumption.

3. Network costs: Data migration consumes valuable data center bandwidth. Therefore, we seek high values of Migration Utility (Section 3.1.1) to ensure that amount of reclaimed space per unit of data transfer is maximized.
4. Scalability: A practical solution should be fast and require little resources (CPU, memory) to process large volumes.

4.5 Results and Inferences

1. Flexibility: We have tested with a wide range of space reclamation objectives (1% to 50%). Higher values (> 50%) may be addressed by finding the complement sets and migrating them. In Figures 2 and 3, the missing values are due to MinimumHash and MinHashApprox failing to address many space reclamation values (e.g. 20% and 30% for VMDK). These techniques produced very skewed clusters for datasets with large files such as VMDK. A first level partitioning (using $hash(f)\%K$) does not suffice, and more intelligent processing is required to further subdivide these partitions to the required size. In contrast, Rangoli addresses all the tested space reclamation values, because the partitioning logic takes the size of the partitions into account.
2. Impact on space consumption: From Figure 2 we see that PSB is least for Rangoli consistently. Naive-du becomes progressively worse for higher space reclamation values. This is because it can easily find small sets with a few unique files. While forming larger sets it does not account for the disk sharing relationships across them and PSB suffers. MinimumHash and MinHashApprox also lead to a significant increase in PSB for some datasets (e.g. see 30% space reclamation for Debian). These strategies try to produce a single hash or a statistical summary of the whole object, that tends to get more inaccurate as the object size increases. In contrast, PSB is consistently less than 2 for Rangoli as it explicitly accounts for block level sharings across files.
3. Network costs: As shown in Figure 3, Rangoli identifies migration bins with MU close to 1 and is consistently better than alternate strategies. This indicates maximum gain (space reclaimed) for the work done (bytes transferred). MU varies widely for alternate strategies as they do not account for the disk sharing relationships between them.
4. Scalability: The detailed running times of Rangoli for

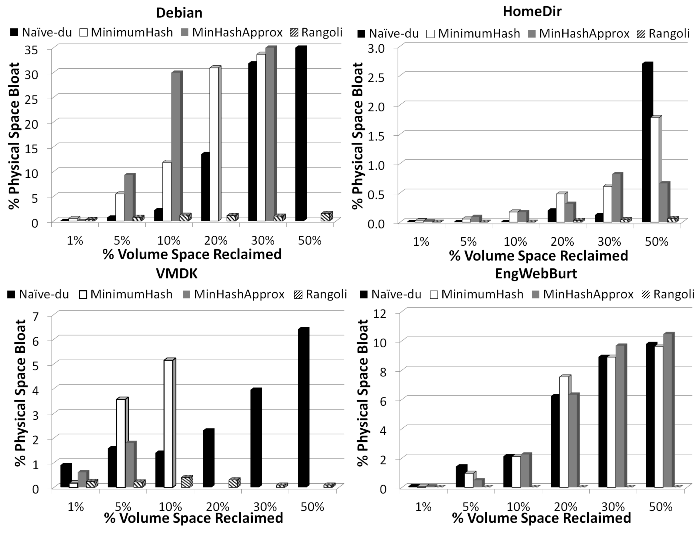

Figure 2: Increase in physical space consumption after space reclamation. (Lower is better)

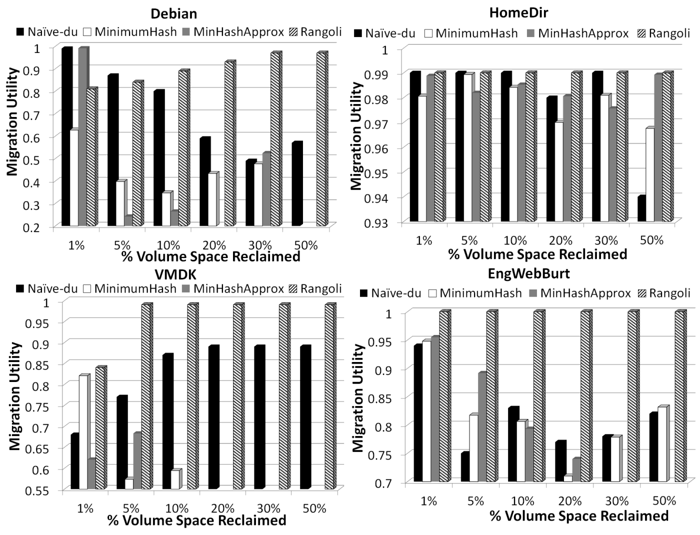

Figure 3: Measure of space reclaimed per byte transfer across the network. (Higher is better)

the different stages and the different datasets are reported in Table 2, which further illustrate the scalability of the algorithms. In the last columns, we present the total running times with 4 parallel threads processing 4 parts of the FPDB (first step). The migration binning and qualification are fast steps and always run in a single threaded mode. We infer that Rangoli is scalable, as even for large real world datasets (EngWebBurt, VMDK) it finished in less than 15 minutes and took less than 64MB of RAM. With synthetic datasets that are even larger, the total running times are still within 30 minutes in parallel mode.

5. CONCLUSION AND FUTURE WORK

In this paper we present a study of challenges with space reclamation in deduped environments. We have shown the limitations of common similarity detection techniques and presented a practical source centric tool for space reclamation. In future, we intend to study destination aware strategies and evaluate the tradeoffs. Presently, Rangoli is implemented as a prototypical tool outside the storage system and we are now working on incorporating it within the storage system. This may bring out other system specific challenges. Also a native implementation in C or C++ may further decrease the running times of these algorithms.

6. ACKNOWLEDGMENTS

We would like to thank Gaurav Makkar for his guidance during the initial stages of this work. We also want to thank Kaladhar Voruganti, Kiran Srinivasan, Parag Deshmukh and our anonymous reviewers for the several insights and valuable feedback.

7. REFERENCES

[1] C. Alvarez. Netapp deduplication for fas and v-series deployment and implementation guide. *Technical ReportTR-3505*, 2011.

[2] G. Appaji Nag Yasa and P. Nagesh. Space savings and design considerations in variable length deduplication. *ACM SIGOPS Operating Systems Review*, 46(3):57–64, 2012.

[3] D. Bhagwat, K. Eshghi, and P. Mehra. Content-based document routing and index partitioning for scalable similarity-based searches in a large corpus. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 105–112. ACM, 2007.

[4] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane. Tradeoffs in scalable data routing for deduplication clusters. In *Proceedings of the 9th USENIX conference on File and stroage technologies*, pages 2–2. USENIX Association, 2011.

[5] G. Forman, K. Eshghi, and J. Suermondt. Efficient detection of large-scale redundancy in enterprise file systems. *ACM SIGOPS Operating Systems Review*, 43(1):84–91, 2009.

[6] K. Jin and E. L. Miller. The effectiveness of deduplication on virtual machine disk images. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, page 7. ACM, 2009.

[7] A. Kathpal, M. John, and G. Makkar. Distributed duplicate detection in post-process data de-duplication.

[8] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble. Sparse indexing: large scale, inline deduplication using sampling and locality. In *Proccedings of the 7th conference on File and storage technologies*, pages 111–123, 2009.

[9] N. Mandagere, P. Zhou, M. A. Smith, and S. Uttamchandani. Demystifying data deduplication. In *Proceedings of the ACM/IFIP/USENIX Middleware'08 Conference Companion*, pages 12–17. ACM, 2008.

[10] D. T. Meyer and W. J. Bolosky. A study of practical deduplication. *ACM Transactions on Storage (TOS)*, 7(4):14, 2012.

[11] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM (JACM)*, 22(2):215–225, 1975.

[12] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320. USENIX Association, 2006.

[13] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou. Scalable performance of the panasas parallel file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, volume 2, pages 1–2, 2008.

[14] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, volume 18, 2008.