Low-Profile Source-side Deduplication for Virtual Machine Backup

Daniel Agun*, Tao Yang*, and Wei Zhang†
* University of California at Santa Barbara †Pure Storage Inc.

Abstract

This paper presents a source-side backup scheme with low-resource usage through collaborative deduplication and approximated lazy deletion when frequent virtual machine snapshot backup is required in a large-scale cloud cluster. The key ideas are to orchestrate multi-round duplicate detection batches among machines in a partitioned asynchronous manner and remove most unreferenced content chunks with approximated snapshot deletion. This paper discusses the challenges, main design and strategies, and evaluation results.

1 Introduction

Frequent backup of virtual machine (VM) snapshots increases the reliability of VMs hosted in a cloud. For example, Aliyun [1], the largest cloud service provider by Alibaba in China, intends to provide automatic frequent backup of all VM images [17]. The challenge is the sheer size of backup data to be transmitted and stored. Source-side deduplication [15, 14, 7] eliminates duplicates before backup data is transmitted; however, its computing resource usage can impact other co-located cloud services. Using a simple dirty-bit method to detect version [4, 15] can identify duplicates among snapshots [15, 14, 7]. Still there is a large amount of network transmission for sending undeduped data. For example, our experimental data with Alibaba datasets show that after dirty bit detection each VM can still send over 24% of raw data chunks to the back end storage. In a cluster with 100,000 VMs with average size of VM snapshot size as 40GB, the total amount of dirty data sent over the network could exceed 0.96 petabyte. Backup products such as ones from NetApp or EMC typically deploy advanced deduplication techniques [10, 19, 9, 12] at the target side. These techniques are memory-intensive even with optimization or approximation [8, 5, 2, 6] and are not ideal for source-level deduplication.

Since backup is a secondary service, cloud providers normally do not want it to contend for resources with other collocated primary services, and it is an open problem how to exploit aggressive source-side deduplication without impacting other collocated cloud services. The focus of this paper is to address this challenge for scalable cloud VM backup. Most previous work on deduplication [19, 2, 5, 9, 8, 12, 17, 16] is an inline approach which uses relatively extensive resources (e.g. a few gigabytes of memory per machine) to optimize the performance of each individual chunk backup operation. The trade-off we play is to develop a low-cost deduplication that optimizes the average backup time of a VM snapshot instead of individual chunk backup operations.

Our previous work on source-side deduplication uses multi-stage synchronous processing [18] with a key disadvantage is that multi-stage synchronization is vulnerable when some participating machine is abnormally slow. The slowness can be caused by a failure or load imbalance due to uneven VM image size distribution. For example, the VM size distribution of production clusters at Alibaba is highly skewed. Figure 1 shows two such clusters, with the left and right clusters having 4200 and 8000 VMs respectively, and skews (Max/Average VM size) of 20 and 45. The deduplication cost in [16] depends on the the shared index size and as the number of VMs increase, the solution can become expensive in terms of memory and CPU usage. Another key weakness in [18, 16] is that snapshot deletion is not addressed. When deleting unused snapshots, a grouped mark-and-sweep approach [8] is effective, but still carries a significant cost in a distributed setting (see Section 5). To provide a full low-cost solution, this paper also addresses the scalable deletion challenge.

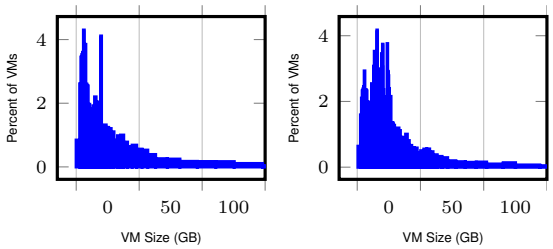

Figure 1: Skewed VM size distribution in two production clusters

2 Strategies and Architecture

Figure 2 illustrates a cloud cluster platform targeted by our scheme. Each server hosts multiple VMs and the co-located backup service collects and fulfills snapshot backup requests for VMs every day. The backup data can be transmitted to a separate secondary storage or to a distributed storage system in this cluster. While VM data is stored in a distributed file system, each machine typically caches actively used virtual images; the backup service can exploit the cached copy so that reading modified backup data can be conducted locally.

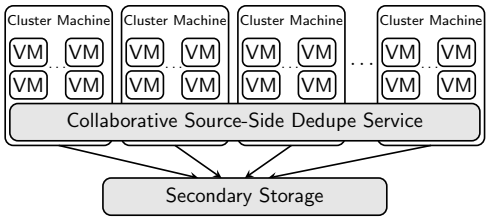

Figure 2: Targeted cloud cluster architecture

Strategies. For inner-VM deduplication, each machine can conduct local deduplication using the dirty-bit method. For the remaining duplicates, cross-VM deduplication is necessary and we adopt the VM-centric strategy [16] which simplifies the cross-VM deduplication by focusing on top-k popular duplicates. This is because global deduplication that detects the appearance of a chunk in any VM requires a substantial resource for cross-machine fingerprint comparison. Our experimental results show that choosing the top 2-4% most popular items (called PDS) for cross-VM deduplication can accomplish close to 98% of deduplication efficiency compared to full deduplication.

We propose two new strategies to accomplish aggressive source-side deduplication with minimal resource usage. 1) Distribute the signature index such as PDS index to a cluster of cloud machines and compare the signatures of candidate chunks with distributed index asynchronously in a multi-round collaborative manner. The asynchronous elimination is necessary to better tolerate load imbalance and straggling tasks. Such a scheme requires a significant amount of buffer space and also some stragglers slow down the entire process and increase the average VM backup time. We play a trade-off through multi-round batch scheduling to limit the size of buffer memory usage and detect the stragglers more aggressively while still allowing a good load balancing.

2) Snapshot deletions can occur frequently since old snapshots become less useful. To filter out unreferenced data from shared duplicates, it would require either maintaining expensive live schemas or conducting global reference counting [8, 13]. We propose an approximation strategy and take advantages of separating popular data chunks from unpopular ones to minimize resource utilization. For chunks used by different snapshots within the same VM, we use a bloom-filter for approximated reference counting with periodic repair. For chunks that are shared among VMs as popular items, we delay reference counting as much as possible, assuming other VMs still use such chunks.

Software architecture. Fingerprint index for popular chunks is partitioned and distributed among the cloud machines that participate in collaborative deduplication. Each physical machine that hosts a VM reads dirty chunks, performs inner-VM duplicate detection, and then sends the signatures of the remaining dirty data to other machines that host popular signature partitions. Thus each machine that hosts a VM and a deduplication service runs the two agents asynchronously. The **backup agent** at each machine leverages the dirty-bit change tracking and inner VM deduplication, and then runs three concurrent threads. The request thread schedules the backup in batches, and initiates duplicate detection requests for each scheduled VM. It reads the dirty documents, divides them into chunks, and computes chunk fingerprints [11]. Then it sends a duplicate detection request for each chunk to a proper machine. The second thread is to accumulate responses from duplicate detection agents, and the third thread performs the real backup of non-duplicate chunks. The **duplicate detection agent** manages three threads to accumulate detection requests and compares them to local index data periodically. It also updates the index when new fingerprints are added the system. The main thread performs multiple rounds of fingerprint comparisons and the multi-round setting provides a flexibility to handle the skewed workload so that small VM data backup can be handled as early as possible.

When some machines fail or respond slowly, a backup agent sets up a timeout and activates a detection task in another machine or temporarily considers these unprocessed requests as non-duplicates. The system conducts global cleanup and removes undetected duplicates periodically (e.g. every few months).

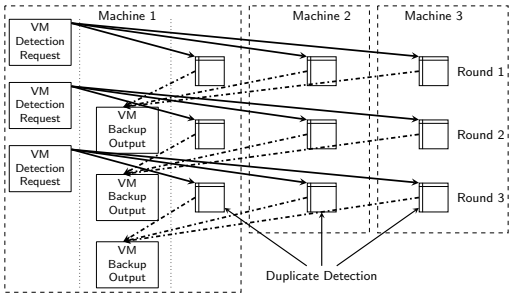

Figure 3: 3-round comparison batches.

3 Scheduling and Resource Control

Each backup agent conducts k rounds of backup batches and it selects the request initiation of a VM with a smaller data size first. The objective is to complete the backup of small VMs first and to shorten the average backup time per VM. Figure 3 shows an example of scheduling in which 3 rounds of fingerprint comparison are triggered at each machine. To control memory usage, we divide the entire fingerprint index evenly and distribute to p machines. For simplicity, we assume each machine hosts VMs and also participates in collaborative deduplication. Each machine further divides the local index to q partitions so that the duplicate detection

agent loads one index partition at a time during comparison with a small memory need. The main memory usage is buffering for communication among machines. 1) Each backup agent uses p request send buffers and $\frac{V}{p*k}$ response receive buffers per machine where V is the total number of VMs hosted in a cluster; 2) Each duplicate detection agent uses p request receive buffers and p send buffers. The fingerprint comparison thread needs memory to load one of q on-disk index partitions and the requests for that partition. The total memory usage is about $\frac{D*V}{r*p}[\frac{1}{k*q} + \frac{b\mu}{q} + \frac{1}{k}]$ where D is the amount of modified data per VM that needs backup after inner VM deduplication; μ is the percentage of unique chunk entries among dirty data accumulated in all snapshots; b is the average number of snapshot versions maintained for each VM; r is the ratio of average chunk size over the index entry size. When the overall index size increases, the memory cost can still be well controlled by increasing q value. This is a key advantage over [16].

Large k value reduces the overall space significantly, but it increases the chance of load imbalance, the cost of synchronization, and also the cost of disk I/O in repeated reading of signature index data for k-round comparisons. Our rule is to allocate less than 100MB of memory usage per machine. The above formula then leads to the estimation of k. For our test data with k=12 which means 9% of VM is handled at a time for each machine.

When a VM is extremely large, special handling is needed to control memory usage. The backup agent divides this VM into a number of sub-VMs and the size of each sub-VM is the same as the average VM size in the cluster. The response accumulation thread buffers the detection response for a chunk based on its corresponding sub-VM ID. The backup data output thread reads one sub-VM at a time for this VM, and checks duplicate status from the corresponding sub-VM response buffer. The metadata for the VM can then be reconstructed by appending each sub-VM's metadata.

For local disk storage usage, cost for buffering messages and storing distributed fingerprint index is relatively small. In our tested dataset, the total disk overhead is under 10GB per machine. The CPU usage is also small, less than 15% of one core during our experiments. The main resource usage in our asynchronous scheme that may create a resource contention is memory and disk I/O bandwidth. For disk I/O bandwidth usage, we set a bandwidth limit with I/O throttling so that other cloud services are minimally impacted. In our experiment, the limit is 60MB/s, which is about 20% of the peak local storage bandwidth.

4 Approximated Lazy Snapshot Deletion

For chunks that are shared among VMs as popular items, we do not need to remove them if other VMs still use the chunk. The system periodically gathers usage information in each index partition, recomputes the top popular items, and adjusts the deletion decisions since each machine maintains the usage information of each chunk per partition. For chunks which are not shared by other VMs, we need to quickly identify if they are used by other snapshots of the same VM. Since the VM size is highly skewed in practice, a large VM may still require a substantial amount of memory for the mark-and-sweep process of data chunks used by all snapshots of this VM. We use a Bloom filter per snapshot to quickly check if the chunks are still referenced by living snapshots. The approximate deletion algorithm contains three aspects.

Computation for snapshot reference summary. Every time there is a new snapshot created, we compute a Bloom-filter with z bits as the reference summary vector for all non-popular chunks used in this snapshot. The items we put into the summary vector are all the references appearing in the metadata of the snapshot.

Fast approximate deletion with summary comparison. To approximately identify if chunks are still used by other undeleted snapshots, we compare the reference of deleted chunks with the merged reference summary vectors of other live snapshots. The merging of live snapshot Bloom-filter vectors uses the bit-wise OR operator. Since the number of live snapshots h is limited for each VM, the time and memory cost of this comparison is small, linear to the number of chunks to be deleted. If a chunk's reference is not found in the merged summary vector, this chunk is not used by any live snapshots. Thus it can be deleted safely. However, among all the chunks to be deleted, there are a small percentage of unused chunks which are misjudged as being in use, resulting in storage leakage.

One advantage of the above fast method is that it can finish and free storage usage immediately, while other off-line methods (e.g. [8, 3]) can't. That is important for storage accounting as users pay for used storage and delayed deletion affects the accounting.

Periodic repair of leakage. Leakage repair is conducted periodically to fix the above approximation error by comparing the live chunks for each VM with what are truly used in snapshot recipes. Since it is a VM-specific procedure, the space cost is proportional to the number of chunks within each VM. This is much less expensive than the VM-oblivious mark-and-sweep which scans snapshot chunks from all VMs, even with optimization [8]. We have conducted an analysis to estimate on how often leak repair should be conducted. Assume that a VM keeps h snapshots in backup storage, creates and deletes one snapshot every day. Let u be the number of chunks brought by initial backup for a VM, Δu be the average number of additional chunks added from one version to next snapshot version. ϵ is the misjudg-

ment rate of being in use caused by the merged Bloom filter. Each Bloom filter vector has z bits for each snapshot and let j be the number of hash functions used by the Bloom filter. Then $\epsilon = (1 - (1 - \frac{1}{z})^{jU})^j$ where $U = u + (h-1)\Delta u$. The number of snapshot deletions when reaching a storage leakage ratio τ can be derived as: $\frac{\tau}{\epsilon} \times \frac{u+(h-1)\Delta u}{\Delta u}$. For the test data in Section 5, $h = 10$, $\Delta u/u = 2.5\%$. To control the leakage under the desired threshold $\tau = 0.1$, leak repair is needed every 19.6 days following the above formula.

5 Evaluation

We have evaluated our prototype implementation on a Linux cluster with 8-core 3.1GHz AMD FX-8120using a production dataset from Alibaba Aliyun's cloud platform [1]. There are 2500 VMs running on 100 physical machines, each machine hosts up to 25 VMs, and each VM keeps 10 snapshots in backup storage. Each VM has about 40GB of storage data on average. The fingerprint for variable-sized chunks is computed using their SHA-1 hash. We have also included some synthetic traces based on VM size distributions from larger Alibaba clusters (see Figure 1). We compare three source-side deduplication schemes. 1) Pure dirty-bit detection. All data are divided into 2MB fix-sized segments and only dirty segments are sent to backup storage. 2) Synchronous multi-stage scheme [18]. 3) Asynchronous scheme.

Resource usage. Table 1 compares resource consumption. Column 2 shows the memory required during deduplication and backup per physical machine. Fingerprint comparison does need more memory than the pure dirty-bit method. The multi-round collaborative scheme uses concurrent thread processing and thus requires more memory than the synchronous scheme. However we limit its memory usage to 90MB, which is a small fraction of the available memory on a server. Column 3 of Table 1 shows the total size of local disk IO required. Our scheme reads backup data twice, thus doubling the I/O size. The collaborative scheme does incur slightly more I/O than the synchronous one because of k rounds of fingerprint comparison. Column 4 lists the final size of output data sent to the backup storage for 2500 VMs with $p = 100$. The dirty-bit method reduces the data size by 75.86%. The final data after our collaborative deduplication is 4.55x smaller. Column 5 shows the network communication size including backup data transmission, and inter-machine deduplication message exchange. The pure dirty-bit approach does not have inter-machine deduplication, but communicates 4x more data because of more undetected duplicates.

Processing time. Table 2 shows the total job time (job span in hours) in Column 2 for a synthetic 2500 VM dataset with a skewed VM size distribution (max/average=20) following Figure 1. Columns 3 is the

Algorithm	Mem (MB)	Local IO (GB)	Storage (GB)	Network (GB)
Dirty Bit	<10	220	22000	22000
Synchronous	40	453	4840	5500
Collaborative	90	491	4840	5500

Table 1: Resource usage comparison per snapshot. Local disk IO and memory costs are per machine. Storage and network cost are for 100 physical machines after deduplication.

Hours	Job span	Backup time per VM (even)	Backup time per VM (skew)
Dirty Bit	1.25	0.05	0.05
Synchronous	50.40	2.75	50.40
Collaborative	2.36	0.23	0.23

Table 2: Job span and average per-VM backup time

average per-VM backup time for this dataset with a relatively even size distribution. Column 4 shows the per-VM time when VM size is skewed. Our scheme reads VM data twice, and thus doubles the job span. Collaborative processing uses 12 rounds and is much faster than the synchronous scheme. In the skewed case, backup time per VM is very high for the synchronous scheme because all VMs must wait for the completion of large VMs at each synchronized stage.

Impact of k rounds. Figure 4 shows the average backup time per VM and job span in the asynchronous scheme. As k increases, the average backup reduces because a large k value provides more opportunities for earlier VM output and the job span increases slightly because more multi-round processing overhead.

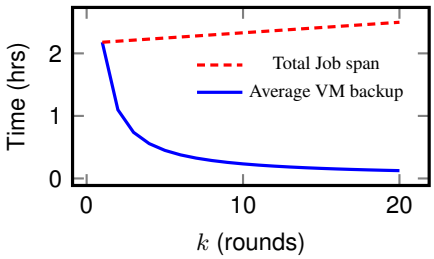

Figure 4: Job span and average per-VM backup time

Effectiveness of Approximate Deletion. Table 3 lists a comparison of processing time and memory usage using the four deletion methods when the number of physical machines $p = 100$ and $p = 50$. These four methods are 1) the standard mark-and-sweep method. 2) Grouped mark-and-sweep [8]. 3) local without using summary vectors (Row 4). 4) approximate local with summary vectors. Last row of Table 3 is the cost of the leakage repair for local with summary vectors. The mark-and-sweep process requires all machines read snapshot metadata for usage comparison. The I/O read speed for the backend distributed file system is about 50MB/second and there is some throughput contention when all machines read data simultaneously: the speed drops to about 30MB/second when $p = 50$ and 25MB/s with

	Time p=50 (hours)	Time p=100 (hours)	Memory (GB)
Mark&sweep	35.9	84.3	1.2–3
Grouped mark&sweep	18.6	43.6	1.2–3
Local w/o sum.	0.7	0.82	0.05 – 1.96
Approx. local	0.012	0.014	0.015
Leak repair	0.7	0.82	0.05 – 1.96

Table 3: Processing time and per-machine memory usage of four deletion methods

$p = 100$. We explain the results for $p = 100$ below. The explanation for $p = 50$ is similar and the result difference for $p = 100$ and $p = 50$ shows our deletion method scales well when p increases.

For the mark-and-sweep method (Row 2 of Table 3) on $p = 100$, we conduct p phases of the mark-and-sweep process. At each phase, a physical machine reads $1/p$ of the non-deduplicated chunk metadata and keeps a reference table in the memory. Then all machines read the meta data of snapshots in parallel and mark the used chunks in the above reference table. The above phase is repeated 100 times (one for for each physical machine). The memory allocated at each physical machine is for the chunk reference table at each phase. the average size is 1.2GB and the maximum is 3GB due to data skew. There is a trade-off between memory usage of a reference table in terms of the size and the total processing time. If we reduce the size of the reference table at each phase, then there are more phases to mark all data and the whole process will take more time. For the grouped mark-and-sweep (Row 3), about 50% of snapshot metadata reading can be avoided by actively tracking the reference usage of non-duplicate chunks. Thus it takes 50% less time, which is about 43 hours, but the memory requirement does not decrease. Notice that in our setting because snapshot deletion occurs frequently, the grouped mark-and-sweep approach becomes less effective in reducing metadata I/O.

For the local deletion without summary vectors (Row 4 of Table 3), all physical machines conduct the mark-and-sweep process in parallel, but each machine only handles one VM at time and the scope of meta data comparison is controlled within the single VM. Popular chunks are excluded. The average memory usage is the index size of non-deduplicated VM chunks, which is about 50MB on average and the largest size is 1.96GB. For approximate deletion with summary vectors (Row 5), each physical machine loads the VM snapshots and only needs to compare with the summary vectors. The memory usage is controlled around 15MB for hosting the summary vectors and small buffers. The deletion time is reduced to less than 1 minute. The periodic leakage repair (Row 6) still takes about 0.83 hours while using an average of 50MB memory. For few big VMs due to data skew, their repair uses upto 1.9GB memory and lasts about 1 minute. Such a repair does not occur often (e.g. every 19.6 days).

Deletions	1	3	5	7	9
Estimated	.02%	.06%	.10%	.14%	.18%
Measured	.01%	.055%	.09%	.12%	.15%

Table 4: Accumulated storage leakage by approximate snapshot deletions ($\Delta u/u = 0.025$)

Table 4 shows percentage of unused storage space per VM misjudged as "in use" due to approximate deletion. In this experiment, we select 105 VMs and let all the VMs accumulate 10 snapshot versions, then start to delete those snapshots one by one in reverse order. Row 1 in Table 4 is the number of snapshot versions deleted. Entry value 3 in this row means that snapshot versions of all VMs from 1 to 3 are deleted. Row 2 is based on a predicted leakage analysis briefly discussed in Section 4. given $\Delta u/u = 0.025$, while row 3 lists the actual average leakage measured during the experiment for all the VMs. The Bloom filter setting is based on $\Delta u/u = 0.025$. After 9 snapshot deletions, the actual leakage ratio reaches 0.0015 and this means that there is only 1.5MB space leaked for every 1GB of stored data. The actual leakage can reach 4.1% after 245 snapshot deletions for all VMs. This experiment shows that the leakage of our approximate snapshot deletion is very small, below the estimated number.

6 Concluding Remarks

The contribution of this work is a scalable solution with multi-round source-side deduplication and approximated deletion for frequent VM snapshot backup. For the tested dataset, the network cost is reduced by 4x and storage cost is reduced by 4.55x compared to a pure dirty-bit-based method. The multi-round deduplication is an order of magnitude faster than a synchronous scheme, when some machines are very slow or have a skewed load. Approximate snapshot deletion only requires 15MB per machine within 1 minute in the tested cases, which is over 3114x faster than the grouped mark-and-sweep method. Leakage repair is 53x faster with 35% to 96% less memory usage. If we were handling the case mentioned in Section 1 with 100,000 VMs using 4000 machines, the size of all VM snapshots sent could be reduced from 0.96 petabyte with a dirty-bit method to 196 terabytes while each physical machine uses about 90MB memory, 10GB disk space, and less than 1% of CPU and sends about 6.6GB metadata for low-profile duplication in less than 3 hours.

Acknowledgments. We thank the anonymous referees for their comments. This work is supported in part by NSF IIS-1528041 and IIS-1118106. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] Alibaba Aliyun. http://www.aliyun.com.

[2] D. Bhagwat, K. Eshghi, D. D. E. Long, and M. Lillibridge. Extreme Binning: Scalable, parallel deduplication for chunk-based file backup. In *IEEE MASCOTS '09*, pages 1–9.

[3] F. C. Botelho, P. Shilane, N. Garg, and W. Hsu. Memory efficient sanitization of a deduplicated storage system. In *FAST'13*, pages 81–94. USENIX.

[4] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li. Decentralized deduplication in san cluster file systems. In *ATC'09*. USENIX.

[5] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane. Tradeoffs in scalable data routing for deduplication clusters. In *FAST'11*, pages 2–2. USENIX.

[6] M. Fu, D. Feng, Y. Hua, X. He, Z. Chen, W. Xia, Y. Zhang, and Y. Tan. Design tradeoffs for data deduplication performance in backup workloads. In *Proc. of USENIX FAST 2015*, pages 331–344, 2015.

[7] Y. Fu, H. Jiang, N. Xiao, L. Tian, F. Liu, and L. Xu. Application-aware local-global source deduplication for cloud backup services of personal storage. *IEEE Trans. Parallel Distrib. Syst.*, 25(5):1155–1165, 2014.

[8] F. Guo and P. Efstathopoulos. Building a high-performance deduplication system. In *ATC'11*, pages 25–25. USENIX.

[9] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezis, and P. Camble. Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality. In *FAST'09*, pages 111–123. USENIX.

[10] S. Quinlan and S. Dorward. Venti: A New Approach to Archival Storage. In *FAST'02*, pages 89–101. USENIX.

[11] M. O. Rabin et al. *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[12] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti. idedup: latency-aware, inline data deduplication for primary storage. In *FAST'12*, pages 24–24. USENIX.

[13] P. Strzelczak, E. Adamczyk, U. Herman-Izycka, J. Sakowicz, L. Slusarczyk, J. Wrona, and C. Dubnicki. Concurrent deletion in a distributed content-addressable storage system with global deduplication. In *FAST*, pages 161–174, 2013.

[14] Y. Tan, H. Jiang, D. Feng, L. Tian, Z. Yan, and G. Zhou. SAM: A semantic-aware multi-tiered source de-duplication framework for cloud backup. In *39th International Conference on Parallel Processing, ICPP 2010, San Diego, California, USA, 13-16 September 2010*, pages 614–623, 2010.

[15] M. Vrable, S. Savage, and G. M. Voelker. Cumulus: Filesystem backup to the cloud. In *FAST'09*, pages 225–238. USENIX.

[16] W. Zhang, D. Agun, T. Yang, R. Wolski, and H. Tang. Vm-centric snapshot deduplication for cloud data backup. In *Proc. of 31st Int. Conf. on Massive Storage Systems and Technologies*, 2015.

[17] W. Zhang, H. Tang, H. Jiang, T. Yang, X. Li, and Y. Zeng. Multi-level selective deduplication for vm snapshots in cloud storage. In *IEEE CLOUD'12*, pages 550–557.

[18] W. Zhang, T. Yang, G. Narayanasamy, and H. Tang. Low-cost data deduplication for virtual machine backup in cloud storage. In *HotStorage'13*. USENIX.

[19] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *FAST'08*, pages 1–14. USENIX.