Low-Cost Data Deduplication for Virtual Machine Backup in Cloud Storage

Wei Zhang*, Tao Yang*, Gautham Narayanasamy*, Hong Tang†
* University of California at Santa Barbara, †Alibaba Inc.

Abstract

In a virtualized cloud cluster, frequent snapshot backup of virtual disks improves hosting reliability; however, it takes significant memory resource to perform data duplication in order to remove excessive redundancy among snapshots. This paper presents a low-cost deduplication solution scalable for a large number of virtual machines. The key idea is to separate duplicate detection from the actual storage backup instead of using inline deduplication, and partition global index and detection requests among machines using fingerprint values. Then each machine conducts duplicate detection partition by partition independently with a minimal memory usage. Another optimization is to allocate and control buffer space for exchanging detection requests and duplicate summary among machines. Our evaluation shows that the proposed multi-stage scheme uses a small or modest amount of system resources while delivering a satisfactory backup throughput.

1 Introduction

Periodic archiving of virtual machine (VM) snaps is important for long-term data retention and fault recovery in case of machine failures. For example, daily backup of VM images is conducted automatically at Alibaba which provides the largest public cloud service in China. The cost of frequent backup of VM snapshots is high because of the huge storage demand. This issue has been addressed by storage data deduplication [9, 15] that identifies redundant content duplicates among snapshots. On the other hand, performing deduplication adds significant memory cost for comparison of content fingerprints. Since each physical machine in a cluster hosts many VMs, memory contention happens frequently. Cloud providers often wish that the backup service only consumes small or modest resources with a minimal impact to the existing cloud services. Another challenge for backup with deduplication is that deletion of old snapshot competes computing resource also. That is because data dependence created by duplicate relationship among snapshots adds processing complexity.

Among the three factors - time, cost and deduplication efficiency, one of them has to be compromised for the other two. For instance, if we were building a deduplication system that has a high rate of duplication detection and has a very fast response time, it would need a lot of memory to hold fingerprint index and cache. This leads to a compromise on cost. Our objective is to lower the cost incurred while sustaining the highest de-duplication ratio and a sufficient throughput in dealing with a large number of VM images.

The traditional approach to deduplication for backup is an inline approach which follows a sequence of block reading, duplicate detection, and non-duplicate block write to the backup storage. Our key idea is to first perform parallel duplicate detection for VM content blocks among all machines before performing actual data backup. Each machine accumulates detection requests and then performs detection partition by partition with minimal resource usage. Fingerprint based partitioning allows highly parallel duplicate detection and also simplifies reference counting management. With careful parallelism and buffer management, this multi-stage detection scheme can provide a sufficient throughput for VM backup.

2 Background and Related Work

At a cloud cluster node, each instance of a guest operating system runs on a virtual machine, accessing virtual hard disks represented as virtual disk image files in the host operating system. For VM snapshot backup, file-level semantics are normally not provided. Snapshot operations are taken place at the virtual device driver level, which means no fine-grained file system metadata can be used to determine the changed data. Only raw access information at disk block level are provided.

Backup systems have been developed to use content fingerprints to identify duplicate content [9, 10]. Offline deduplication is used in [5, 2] to remove previously written duplicate blocks during idle time. Several techniques have been proposed to speedup searching of duplicate fingerprints. For example, the data domain method [15] uses an in-memory Bloom filter and a prefetching cache for data blocks which may be accessed. An improvement to this work with parallelization is in [13]. Still such methods use a significant amount of memory resource for filtering and caching. The approximation techniques are studied in [3, 14] to reduce memory requirement with a tradeoff of the reduced deduplication ratio. In comparison, this paper focuses on full deduplication without approximation.

Additional inline deduplication techniques are studied

in [7, 6, 11]. All of the above approaches have focused on such inline duplicate detection in which deduplication of an individual block is on the critical write path. In our work, this constraint is relaxed and there is a waiting time for many duplicate detection requests. This relaxation is acceptable because in our context, finishing the backup of required VM images within a reasonable time window is more important than optimizing individual VM block backup requests.

3 System Design

We consider deduplication in two levels. The first level uses coarse-grain segment dirty bits for version-based detection [4, 12] to identify difference from the previous snapshot to the current snapshot. Our experiment with Alibaba's production dataset shows that over 70 percentage of duplicates can be detected using segment dirty bits when the segment size is 2M bytes. This setting requires OS to keep the metadata for segment dirty bits and the amount of space for such segment dirty bits is negligible. In the second level of deduplication, content blocks of dirty segments at each VM are compared with the fingerprints of unique blocks from the previous snapshots. Our key strategies are explained as follows.

- **Separation of duplicate detection and data backup.** The second level duplicate detection requires a global comparison of content fingerprints. Our approach is to perform duplicate detection first before actual data backup. That requires a prescanning of data blocks in dirty VM segments and after that, real backup for those non-duplicates is conducted. That does incur an extra round of local VM block reading while avoiding inline deduplication and leading to a much smaller resource requirement. During VM reading, detection requests are accumulated on the disk for a period time in a partitioned manner. Aggregated duplicate requests can be processed partition by partition. Since each partition corresponds to a small portion of global content index, the cost to process detection requests within a partition is significantly smaller than processing all the requests globally.
- **Buffered data redistribution in parallel duplicate detection**. Let *global index* be the meta data containing the fingerprint values of unique blocks of snapshots in all VMs and the reference pointers which lead to the location of raw data. A logical way to distribute detection requests among machines is to partition based on fingerprint values of content blocks in a disjointed manner. Initial VM data reading follows the VM distribution among machines and the detected duplicate summary results need to be collected following the VM distribution. Therefore, there are two all-to-all data redistribution operations involved. One is to map detection requests from VM-based distribution to fingerprint based distribution. Another one is to map duplicate summary from fingerprint-based distribution to VM based distribution. The redistributed data needs to be accumulated on the disk to minimize the use of memory. To minimize the excessive disk seek cost, outgoing or incoming data exchange messages need to be buffered to bundle small messages. Given there are $p \times q$ partitions where p is the number of machines and q is the number of fingerprint-based partitions at each machine, space per each buffer is small under the memory constraint for large p or q values. This counteracts the effort of seek cost reduction. We have designed an efficient data exchange and disk data buffering scheme to address this.

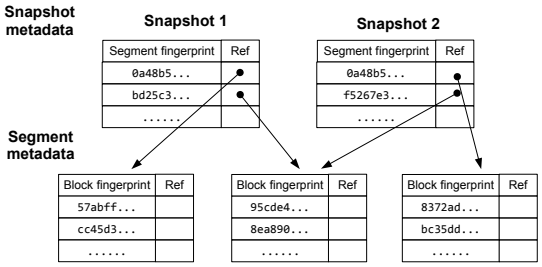

Figure 1: Metadata structure of a VM snapshot.

We assume a flat architecture in which all p machines that host VMs in a cluster can be used in parallel for deduplication. A small amount of local disk space and memory on each machine can be used to store global index and temporary data. The real backup storage can be either a distributed file system built on this cluster or use another external storage system.

The representation of each snapshot in the backup storage has a two-level index structure in the form of a hierarchical directed acyclic graph as shown in Figure 1. A VM image is divided into a set of segments and each segment contains hundreds of content blocks. These blocks are of variable-size, partitioned using the standard chunking technique [8] with 4KB as the average block size. The snapshot metadata contains a list of segments and other meta data information. Segment metadata contains its content block fingerprints and reference pointers. If a segment is not changed from one snapshot to another, indicated by a dirty bit embedded in the virtual disk driver, its segment metadata contains a reference pointer to an earlier segment. For a dirty segment, if one of its blocks is duplicate to another block in the system, the block metadata contains a reference pointer to the earlier block.

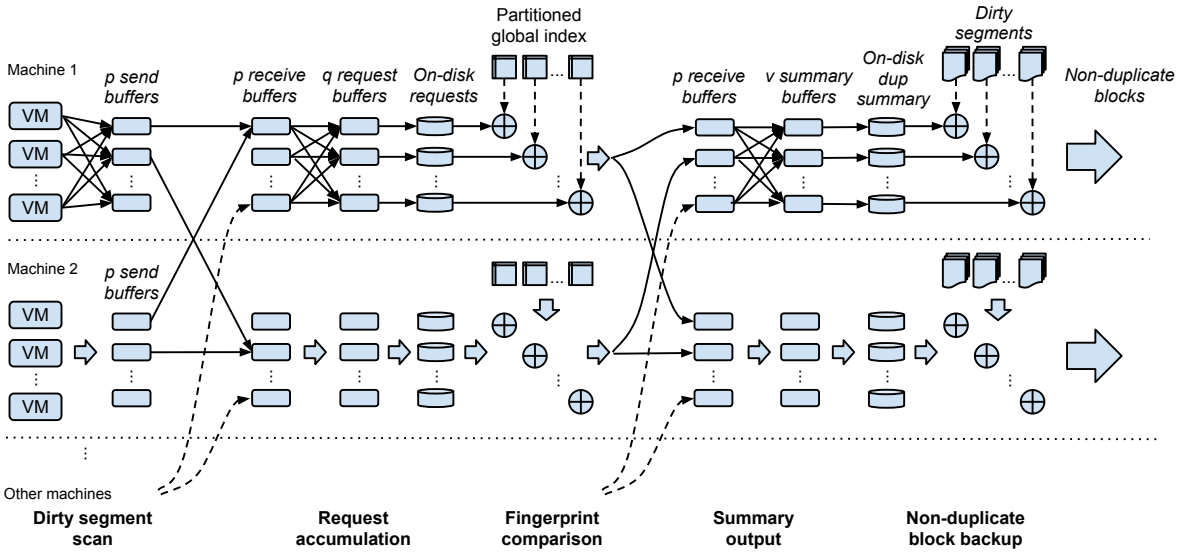

Figure 2: Processing flow of Stage 1 (dirty segment scan and request accumulation), Stage 2 (fingerprint comparison and summary output), and Stage 3 (non-duplicate block backup).

The data flow of our multi-stage duplicate detection is depicted in Figure 2. In Stage 1, each machine independently reads VM images that need a backup and forms duplicate detection requests. For those segments that have been modified since last backup indicated by dirty bits, the system divides each segment into a sequence of chunk blocks, computes the meta information such as chunk fingerprints, sends a request to a proper machine, accumulates received requests into a partition on the local temporary disk storage. The partition mapping is using a hash function applied to the content fingerprint. Assuming all machines have a homogeneous resource configuration, each machine is evenly assigned with q partitions of global index and accumulates corresponding requests on the disk. There are two options to allocate buffers at each machine. 1) Each machine has $p \times q$ send buffers corresponding to $p \times q$ partitions in the cluster since a content block in a VM image of this machine can be sent to any of these partitions. 2) Each machine allocates p send buffers to deliver requests to p machines; it allocates p receive buffers to collect requests from other machines. Then the system copies requests from each of p receive buffers to q local request buffers, and outputs each request buffer to one of the request partitions on the disk when this request buffer becomes full. Option 2 depicted in Figure 2 is much more efficient than Option 1 because $2p + q$ is much smaller than $p \times q$. As a result, each buffer in Option 2 has a bigger size to accumulate requests and that means less disk seek overhead.

Stage 2 is to load disk data and perform fingerprint comparison at each machine one request partition at a time. At each iteration, once in-memory comparison between an index partition and request partition is completed, duplicate summary information for segments of each VM is routed from the fingerprint-based distribution to the VM-based distribution. The summary contains the block ID and the reference pointer for each detected duplicate block. Each machine uses memory space of the request partition as a send buffer with no extra memory requirement. But it needs to allocate p receive buffers to collect duplicate summary from other machines. It also allocates v request buffers to copy duplicate summary from p receive buffers and output to the local disk when request buffers are full.

Stage 3 is to perform real backup one VM at a time. The system loads the duplicate summary of a VM, reads dirty segments of a VM, and outputs non-duplicate blocks to the final backup storage. Additionally, the global index on each machine is updated with the meta data of new chunk blocks. When a segment is not dirty, the system only needs to output the segment meta data such as a reference pointer. There is an option to directly read dirty blocks instead of fetching a dirty segment which can include duplicate blocks. Our experiment shows that it is faster to read dirty segments in the tested workload. Another issue is that during global index update after new block creation in the backup storage, the system may find some blocks with the same fingerprint may be created redundantly. For example, two different VM blocks that have the same fingerprint are not detected because the global index has not contained such a fingerprint yet. The redundancy is discovered and logged during the index update and can be repaired periodically when necessary. Our experience is that there is a redundancy during the initial snapshot backup and

once that is repaired, the percentage of redundant blocks due to concurrent processing is insignificant.

The above steps can be executed by each machine using one thread to minimize the use of CPU, network bandwidth, and disk bandwidth. The disk storage usage on each machine is fairly small for storing part of global index and accumulating duplicate detection requests that contain fingerprint information. We impose a memory limit M allocated for each stage of processing at each machine. The usage of M is controlled as follows and the distribution among buffers is optimized based on the relative ratio between the cross-machine network startup cost and disk access startup cost such as seek time. Slower start cost requires more buffer space.

- For Stage 1, M is divided for 1) an I/O buffer to read dirty segments; 2) $2p$ send/receive buffers and q request buffers.
- For Stage 2, M is divided for 1) space for hosting a global index partition and the corresponding request partition; 2) p receive buffers and v summary buffers.
- For Stage 3, M is divided for 1) an I/O buffer to read dirty segments of a VM and write non-duplicate content blocks to the backup storage. 2) summary of duplicate blocks within dirty segments of a VM.

Snapshot deletion. Each VM will keep a limited number of automatically-saved VM snapshots and thus expired snapshots will be deleted unless its owner explicitly asks for its retention. We adopt the idea of mark-and-sweep [6]. A block or a segment can be deleted if its reference count is zero. To delete useless blocks or segments periodically, we read the meta data of live snapshots and compute the reference count of all blocks and segments in parallel while following the fingerprint-based partitioning. Similar to the multi-stage duplicate detection process, reference counting is conducted in multi-stages. Stage 1 is to read the segment and block metadata of snapshots to accumulate reference count requests in different machines in the fingerprint based distribution. Stage 2 is to count references within each partition and detect those records with zero reference. The backup data repository logs deletion instructions, and will periodically perform a compaction operation when its deletion log is too big.

4 Evaluation

We have implemented and evaluated a prototype of our multi-stage deduplication scheme on a Linux cluster of multi-core AMD Bulldozer FX8120 and Intel Nehalem E5530 machines. Our implementation is based on Alibaba's Xen cloud platform [1, 14]. Objectives of our evaluation are: 1) Analyze the deduplication throughput and effectiveness for a large number of VMs. 2) Examine the impacts of buffering during metadata exchange.

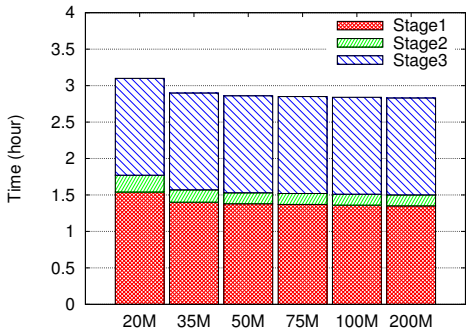

Figure 3: Parallel time when memory limit varies.

We have performed a trace-driven study using a 1323 VM dataset collected from 100 Alibaba Aliyun's cloud nodes [14] and each of machine nodes has 16 cores and 12 disk drives, hosting up to 25 VMs. For each VM, the system keeps 10 automatically-backed snapshots in the storage while a user may instruct extra snapshots to be saved. The backup of VM snapshots is completed within a few hours every night. Based on our study of its production data, each VM has about 40GB of storage data usage on average including OS and user data disk. Each VM image is divided into 2 MB fix-sized segments and each segment is divided into variable-sized content blocks [8] with an average size of 4KB. The signature for variable-sized blocks is computed using their SHA-1 hash.

The seek cost of each random IO request in our test machines is about 10 milliseconds. The average I/O bandwidth of local storage is about 50MB/second in the presence of other I/O jobs. Noted that a RAID 0 configuration can deliver over 300MB/second with 4 drives or more and our setting uses 16.7% or less of local storage bandwidth. The final snapshots are stored in a distributed file system built on the same cluster.

The total local disk usage on each machine is about 8GB for the duplicate detection purpose, mainly for global index. Level 1 segment dirty bits identify 78% of duplicate blocks. For the remaining dirty segments, block-wise full deduplication removes about additional 74.5% of duplicates. The final content copied to the backup storage is reduced by 94.4% in total.

Figure 3 shows the total parallel time in hours to backup 2500 VMs on a 100-node cluster a when the memory limit M imposed on each node varies from 20MB to 200MB. This figure also depicts the time breakdown for Stages 1, 2, and 3. The time in Stages 1 and 3 is dominated by the two scans of dirty VM segments and the time for copying to the final backup storage is overlapped with VM scanning. During dirty

VM segment reading, the average number of consecutive dirty segments is 2.92. The overall processing time does not have a significant reduction as M increases to 200MB. The aggregated deduplication throughput is about 9.6GB per second, which is the size of 2500 VM images divided by the parallel time. The system runs with a single thread and its CPU resource usage is 10-13% of one core. The result shows the backup with multi-stage deduplication for all VM images can be completed in about 2.9 hours with 35MB memory, 8GB disk overhead and a small CPU usage. As we vary the cluster size p, the parallel time does not change much, and the aggregated throughput scales up linearly since the number of VMs is $25p$.

Table 1 shows performance change when memory limit M=35MB is imposed and the number of partitions per machine (q) varies from 50 to 1500. Row 2 is the memory space required to load a partition of global index and detection requests. When $q = 50$, the required memory is 156MB, which is not a viable choice under constraint M =35MB. Row 3 is the parallel time and Row 4 is the aggregated throughput of 100 nodes. Row 5 is the parallel time if we use Option 1 with $p \times q$ send buffers described in Section 3. With the large number of buffers, the available space per buffer reduces significantly as q increases, which leads to a big increase of IO requests and seek cost.

Table 1: Performance when the number of partitions per machine node varies and M=35MB is imposed.

#Partitions	50	275	450	750	1500
Index+request (MB)	156	28.4	17.4	10.7	5.2
Total Time (Hours)	N/A	2.9	2.91	3	3.1
Throughput GB/s	N/A	9.6	9.5	9.3	8.9
Total time (Option 1)	N/A	7.8	11.7	14.8	26

5 Conclusion Remarks

The contribution of this work is a low-cost multi-stage parallel deduplication solution. Because of separation of duplicate detection and actual backup, we are able to evenly distribute fingerprint comparison among clustered machine nodes, and only load one partition at time at each machine for in-memory comparison. The trade-off is that every machine has to read dirty segments twice and that some deduplication requests are delayed for staged processing.

The evaluation shows that the overall deduplication time and throughput of 100 machines are satisfactory with about 9.6GB per second for 2500 VMs. During processing, each machine uses 35MB memory, 8GB disk space, and 10-13% of one CPU core with a single thread execution. Thus the proposed scheme is resource-friendly to the existing cloud services. Our future work is to conduct more experiments with production workloads.

References

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

[2] C. Alvarez. NetApp Deduplication for FAS and V-Series Deployment and Implementation Guide. NetApp. Technical Report TR-3505 , 2011.

[3] 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, 2009.

[4] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li. Decentralized deduplication in san cluster file systems. In *USENIX ATC'09*, 2009.

[5] EMC. Achieving storage efficiency through EMC Celerra data deduplication. White Paper, 2010.

[6] F. Guo and P. Efstathopoulos. Building a high-performance deduplication system. In *USENIX ATC'11*, pages 25–25, 2011.

[7] 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, 2009.

[8] U. Manber. Finding similar files in a large file system. In *USENIX Winter 1994 Technical Conference*, pages 1–10, 1994.

[9] S. Quinlan and S. Dorward. Venti: A New Approach to Archival Storage. In *FAST '02*, pages 89–101, 2002.

[10] S. Rhea, R. Cox, and A. Pesterev. Fast, inexpensive content-addressed storage in foundation. In *USENIX ATC'08*, pages 143–156, Berkeley, CA, USA, 2008. USENIX Association.

[11] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti. idedup: latency-aware, inline data deduplication for primary storage. In *FAST'12*, 2012.

[12] M. Vrable, S. Savage, and G. M. Voelker. Cumulus: Filesystem backup to the cloud. In *FAST'09*, pages 225–238, 2009.

[13] T. Yang, H. Jiang, D. Feng, Z. Niu, K. Zhou, and Y. Wan. Debar: A scalable high-performance de-duplication storage system for backup and archiving. In *IEEE IPDPS*, pages 1–12, 2010.

[14] 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, 2012.

[15] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *FAST'08*, pages 1–14, 2008.