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

Wei Zhang*, Tao Yang*, Gautham Narayanasamy*, Hong Tang†
*Department of Computer Science, 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 and CPU resource to perform data duplication in order to remove excessive redundancy among snapshots. This paper presents a low-cost clustered deduplication solution scalable for handling a large number of virtual machines. The key idea is to separate duplicate detection from the actual storage data backup instead of using inline deduplication, and partition global index and detection requests among machines using fingerprint values. Then each machine conducts duplicate pre-detection partition by partition independently with a minimal memory usage. Another optimization is to allocate and control buffer space for exchanging detection messages and duplicate summary among machines. Our experiments show that the proposed scheme uses a very small 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 [6, 9] 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 the minimal resource usage without impacting any existing cloud service. 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, especially when VMs can migrate around in the cloud.

Among the three factors - time, cost and de-deduplication efficiency, one of them has to be compromised for the other two. For instance, if we were build 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 do fast comparison. This leads to a compromise on cost. The objective of our project is to lower the cost incurred while sustaining the highest de-duplication ratio. At the same time, we ensure the aggregate read/write throughput of the system is sufficiently scalable for dealing with backup of a large number of virtual machine images.

The traditional approach to deduplication for backup is an inline approach which follows a sequence of VM block reading, duplicate detection, and non-duplicate block write to the backup storage. Our key idea is to first perform parallel duplicate pre-detection for VM content blocks among all machines before performing actual data backup. Each machine accumulates detection requests and then performs pre-detection partition by partition with minimal resource usage. Fingerprint based partitioning allows highly parallel duplicate pre-detection and also simplifies reference counting management. With careful parallelism management and message buffering during cross-machine data exchange, this pre-detection scheme can provide a sufficient throughput for 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. Each physical hosts many VMs and a cloud cluster has petabytes of data that need to be backed up at daily basis, if not more frequently. But at the same time snapshot tasks must not affect the normal cloud service, which means that ideally only a very small slice of CPU and memory can be used for the backup purpose.

The previous work for storage backup has extensively used data deduplication techniques can eliminate redundancy globally among different files from different users. Backup systems have been developed to use content hash (finger prints) to identify duplicate con-

tent [6, **?**]. Several techniques have been proposed to speedup searching of duplicate content. For example, Zhu et al. [9] tackle it by using an in-memory Bloom filter and prefetch groups of chunk IDs that are likely to be accessed together with high probability. It takes significant memory resource for filtering and caching. The approximation techniques are studied in Deepavali et al. [2] and Zhang et al. [**?**] 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 [4, 8]. All of the above approaches have focused on such inline duplicate detection, namely given a content chunk block, how to detect if it is duplicate instantly to trigger the data backup of this block or not before the detection request for the next blocks can be launched. In our work, this constraint is relaxed and there is a waiting time for many duplicate detection requests. Our goal is to minimizing computing resource without affecting the overall time of backup.

## 3 Key Ideas and Architecture

We consider deduplication in two levels. The first-level of data deduplication can be accomplished efficiently and inexpensively using a coarse-grain segment-level dirty bit based detection in the OS level to identify version difference from the previous snapshot to the current snapshot. Version-based detection has been used in the previous work [3, **?**, **?**] and we emphasize a segment-level coarse-grain setting to reduce the efforts in detection and maintaining dirty bits. Our experiment with Alibaba"s production dataset shows that over 70 percentage of duplicates can be detected using segment-based dirty bits while the segment size is as large as 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, remaining content blocks of each VM are compared with the signatures of all unique blocks stored in the previous snapshots.

Our key ideas are summarized as follows.

- **Separation of duplicate detection and data backup.** The second level duplicate detection requires a global comparison of content fingerprints of stored chunks with the new chunks scanned during the backup process. Bloom filters and caching allows some of content fingerprints to be stored on the cheap disks [9]. Such an optimization is good for inline deduplication where chunks scanned need to be determined to be duplicate or not instantly without waiting while it memory consumption is still significant in competing with stand virtual machine activities. Our idea is to perform duplicate detection first before actual data backup. That requires a pre-scanning of data blocks of VM images and after that, real backup for those non-duplicates is conducted. That does incur an extra round of I/O reading for non-duplicate blocks while avoiding inline deduplication and leading to a much smaller resource requirement. During duplicate detection phase, detection requests can be 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 requests within a partition is significant smaller than process all the requests globally.
- **Buffered data redistribution in parallel duplicate detection**. We let *global index* be the meta data containing the fingerprint values of all unique blocks in the entire cluster and the reference pointers which lead to the actual location of raw data. A duplicate detection request for a content block needs to be compared with the global index to determine if this block is non-duplicate. If it is a duplicate, return the corresponding reference pointer. We conduct parallel processing of detection requests in all machines in a cloud cluster. A logical way to distribute detection requests is to partition based on the content hashing of data blocks and the global index can also be partitioned accordingly in a disjointed manner. Initial data follows the VM distribution among machines and the detected results represented as duplicate summary need to be collected following VM distribution. Therefore, there are two all-to-all data remapping operations involved. One is to map detection requests from VM-based requests to the fingerprint based distribution. One is to map duplicate summary from fingerprint-based requests to the VM based distribution. The redistributed data needs to be accumulated on the disk to minimize the use of memory. To minimize the cost of disk seek, 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, the number of buffers is big and space per each buffer is small given the memory constraint. Then there will be a large number of storage IO operations involved, suffering the huge seek overhead. We have designed an efficient data exchange and disk data buffering scheme to address this.

We assume a flat architecture in which all $p$ machines that host VMs in a cluster can be used in parallel for deduplication. The non-duplicate snapshot blocks are

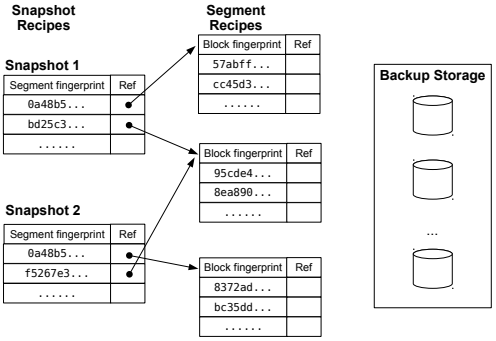


Figure 1: Metadata structure of a VM snapshot.

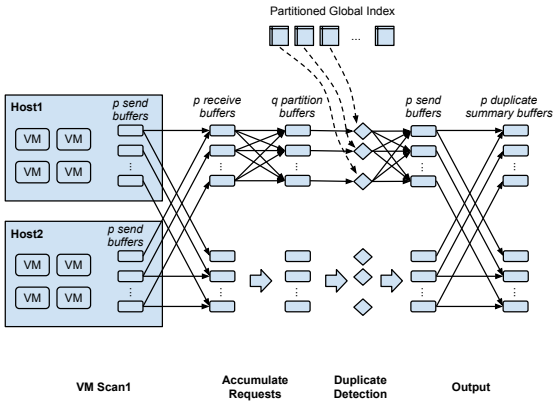


Figure 2: Processing steps of duplicate detection.

stored either in a distributed file system built on this cluster or in another external storage system. We assume that a small amount of local disk space and memory on each machine can be used to store global index and request accumulation.

## 3.1 Data Structure

The representation of each snapshot saved in the backup storage has a two-level level index data structure in the form of a hierarchical directed acyclic graph as shown in Figure 1. An VM image is divided into a set of segments and each segment contains hundreds of content blocks from the bottom level. These blocks are of variable-size, partitioned using the standard chunking technique [5] with 4KB as the average block size.

Segment metadata (called segment recipe) records its content block hashes and data pointers. The snapshot recipe contains a list of segments and other meta data information. If a segment is not changed from one snapshot to another, indicated by a dirty bit embedded in the virtual disk driver, its content blocks are not changed as well, thus its segment recipe contains a reference pointer to an earlier segment. If a block is duplicate to another block in the system, the block recipe contains a reference pointer to an earlier block.

## 3.2 Processing flow and resource usage

The data flow of our partition-based duplicate detection is depicted in Figure 2.

- The first step is that each machine independently reads virtual machine 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 the metadata of a chunk block, and accumulate into a partition bucket on the 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 $q$ parititons of content hash and accumulates corresponding requests on the disk. There are two options to assign buffers in each machine to buffer requests for each partition. 1) Each machine has $p \times q$ send buffers corresponding to $p \times q$ partitions in the cluster since any content block in a VM image of this machine can be sent to any of these partitions. 2) Each machine has $p$ receive buffers correspond to $p$ machines in the cluster and $q$ receive buffers to temporarily accumulate messages from other machines, and then output each buffer to the local disk when it becomes full. Option 2 is much more efficient than Option 1 because $p + q$ is much smaller than $p \times q$, as a result each buffer has a bigger size to accumulate data and results in much less disk seek overhead.
- The second step is to perform duplicate detection for all requests accumulated in the partitions. The system maintains the index of all chunk fingerprints divided in the buckets. We load the global index partition and accumulated corresponding requests, and compare them to identify the duplicated blocks. Then we unload them, load the global index and accumulated requests for another partition. This process is repeated until all buckets are processed.

  As chunk blocks are detected as duplicates, the meta information on duplicates is remapped from fingerprint-based distribution to VM-based distribution. In this way, duplicate information for segments of each VM is collected in one place, which will be used in next step. In the sending side, each machine scans identified duplicates for one target

machine, and then scans for another machine. In this way, it sends duplicate summary to the corresponding machines one by one to minimize the startup cost for sending a message. In the receiving side, each machine allocates $q$ receive buffers corresponding to $q$ local partitions, to hold received messages from other machines. When a receive buffer for a VM is full, the data is written to the local disk storage.

- The third step is to read non-duplicate blocks from each VM and output to the final backup storage. Additionally, the global index on each machine is updated with the meta data of new chunk blocks added. In this process, if a segment is not dirty, we only need to output the segment meta data such as reference pointers to the original segments previously saved in the backup. If a segment is dirty, we use the duplicate summary information from Step 2 for internal duplicate blocks of this dirty segment and output the remaining non-duplicate blocks to the backup storage.

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 limit for memory allocated at each machine node to accomplish low-cost deduplication. Memory usage for each step of above scheme is carefully controlled as follows.

- For Step 1, the allocated memory for each machine is evenly divided for $p$ send buffers and $q$ receive buffers.
- For Step 2, the allocated memory serves 2 purposes. 1) Space for hosting a partition of global index and the corresponding partition of deduplication requests. 2) Buffer space for $q$ receive buffers. The duplicate summary sent other machines and acclimated for each VM contains the block ID and the reference pointer.
- For Step 3 to perform the real backup for each VM, memory is used to hold the meta information summary of all duplicate blocks obtained from Step 2. Another memory usage is to fetch a sequence of blocks from a virtual machines (most of them are non-duplicate) and write them to the backup storage.

**Snapshot deletion.** Each virtual machine will keep at most $x$ snapshots of VM images and thus expired snapshots will be deleted unless its owner explicitly asks for its retention. A block or the entire segment can be deleted if its reference counter is zero. We periodically read all receipts of active snapshots and compute the counter of all blocks while still following the fingerprint-based partitioning. That is based on the idea of mark-and-sweep [**?**] while we deal such snapshot deletion in three partition-based steps similar to the VM snapshot backup. Step 1 is to read the meta data of snapshots such as segment and block recipes and accumulate reference count requests in different machines in the fingerprint based distribution. The second step is to count references within each partition and detect those records with zero reference. The third step is that every VM gathers the confirmed deletion requests for its content and sends them to the backup storage. The archival data repository may log these deletion requests, and will periodically perform a compaction operation when its deletion log is too big.

# 4 Evaluation

We have implemented and evaluated a prototype of the partition-based deduplication scheme on a Linux cluster of AMD Bulldozer FX8120 and Intel Nehalem E5530 machines. Objectives of our experimental evaluation are: 1) Analyze the processing time and throughput of backup for a large cluster of virtual machines. 2) Assess the effectiveness of partition-based duplication detection backup. 3) Examine the impacts of buffering during meta data exchange.

## 4.1 Experimental setup

Our implementation is based on Alibaba's Xen cloud platform [1, **?**]. At Alibaba's Aliyun, the target is to backup cluster of a few thousands nodes with 25 VMs on each machine. Based on the Alibaba data studied, each VM has about 40GB of storage data usage on average including OS and user data disk. The backup of VM snapshots is completed within a few hours every night, For each VM, the system keeps 10 automatically-backed snapshots in the storage while a user may instruct extra snapshots to be saved.

We have performed a trace-driven study using a dataset containing 10 snapshots of 35 VMs. That is part of a 1323 VM dataset collected from 100 Aliyun's cloud nodes in our earlier work with Alibaba Aliyun [**?**]. All data are divided into 2 MB fix-sized segments and each segment is divided into variable-sized content blocks [5, 7] with an average size of 4KB. The signature for variable-sized blocks is computed using their SHA-1 hash.

```
memory
#partitions/node 50 100 250 275 300
450 500 750 1500 2000
Memory meta data 0.161 0.0805 0.0322
0.029272727 0.026833333 0.017888889
```

```
0.0161 0.010733333 0.005366667
0.004025
Global index 0.115 0.0575 0.023
0.020909091 0.019166667 0.012777778
0.0115 0.007666667 0.003833333
0.002875
Seek (Step 1) 0.0212962960.0425 0.106 0.212962963 0.425925926
0.191666667 0.212962963 0.319444444
0.638888889 0.851851852
Seek Option 2: 6.388888889
12.77777778 31.94444444 35.13888889
38.33333333 57.5 63.88888889
95.83333333 191.6666667 255.5555556
Total time 3.059031718 3.078670034
3.131061779 3.137163671 3.141047877
3.313871832 3.296994302 3.377385758
3.689876824 3.901831421
throughput  per node 0.090805785
0.090226551 0.088716799 0.088544242
0.088434748 0.083822728 0.084251822
0.082246387 0.075281044 0.07119164
```

Table **??** shows the performance of partition-based deduplication when the number of parititons per machine ($q$) varies from 50 to 2000. Row 2 shows the memory usage needed to load a partition of global index and detection requests, varying from 161MB to 4.025MB. As we impose a constraint of 30MB, then only $q > 300$ would be a viable choice. Row 3 shows the majority of memory usage is for global index. Row 4 is the total time for a 100-node cluster. Row 5 is the throughput per machine reported is the total amount of virtual machine images divided by the processing time. The processing time includes the first scanning time of dirty VM segments to generate block fingerprints and the second scan time for real backup.

```
q= is chosen between 500 or 100
Memory limit:   20 30M   50M  75M  100M  200M
Time:          3.2  3.11   3  2.99 2.96  2.93
Time for two scans: 2.78 2.78 2.78 2.78 2.78
```

Table **??** shows the performance of partition-based deduplication when the memory limit imposed on each node, varying from The overall processing time does not have a significant reduction as we increase memory usage 20MB to 200MB. As Row 2 shows, the time is mainly dominated by the IO costs of two scans of VMs. Thus we conclude that using about 30MB is sufficient to have VM backup completed in parallel within about 3.11 hours.

We have also compared our pre-detection approach with another approach by combining dirtybit segment detection and the data domain method [**?**]. The bloomer filter setting results a 1:10 ratio of index reduction for in-memory search before visiting the disk for the full index with 2% false positives. The prefetching cache hit ratio is set to be 98.96% based a number in [**?**]. We measure its total time including one scan of VM dirty segments in about 1.38 hours. That approach takes about 3.19 hours comparable with our pre-detection scheme while its memory cost includes 1GB of bloom filters and additional space for cache. That is an significant amount of space, competing with other cloud service while memory cost of our setting is insignificant.

## 5 Conclusion

The contributing of this work is a low-cost and scalable clustered deduplication solution using a non-inline approach. 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. That leads to an insignificant system resource, which does not compete with the existing cloud services. The tradeoff is that every machine has to read virtual machine images twice to accomplish backup. When the cluster size changes, our experiment also shows a linear speedup of overall throughputs because highly parallel fingerprint comparisons. The cluster of 100 nodes can deliver about 8.8GB per second deduplication performance. We currently assume each machine performs backup for all VMs hosted and in practice, the system only needs to backup active VMs and thus the overall backup time would actually be much smaller. Our future work is to conduct more scalability studies and experiments with more production trace data.

Our experiments show that the proposed scheme uses a very small amount of system resources while accomplishing a satisfactory backup throughput in a large cloud setting.

## References


[1] Aliyun Inc. 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 *Modeling, Analysis Simulation of Computer and Telecommunication Systems, 2009. MASCOTS '09. IEEE International Symposium on*, pages 1–9, 2009.

[3] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li. Decentralized deduplication in SAN cluster file systems. page 8, June 2009.

[4] 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*, pages 111–123, 2009.

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

[6] S. Quinlan and S. Dorward. Venti: A New Approach to Archival Storage. In *FAST '02: Proceedings of the Conference on File and Storage Technologies*, pages 89–101, Berkeley, CA, USA, 2002. USENIX Association.

[7] M. O. Rabin. Fingerprinting by random polynomials. Technical Report TR-CSE-03-01, Center for Research in Computing Technology, Harvard University, 1981.

[8] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti. idedup: latency-aware, inline data deduplication for primary storage. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, FAST'12, 2012.

[9] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, pages 1–14, Berkeley, CA, USA, 2008. USENIX Association.