Parallel Deduplication with Batch Scheduling for Collocated Virtual Machine Backup in the Cloud

Daniel Agun*, Wei Zhang*, and Tao Yang*
* University of California at Santa Barbara

Abstract

In a virtualized cloud cluster, collocation of backup storage service reduces hosting cost and network traffic caused by frequent virtual machine backup. However, deduplication can be expensive and its resource usage affects other collocated services. This paper presents a low-cost and parallel deduplication with batch scheduling to orchestrate backup activities of clustered servers with minimum resource usage. The key idea is to separate duplicate detection from the actual storage backup, and partition global index and detection requests among machines using fingerprint values. Then each machine conducts duplicate detection partition by partition independently. Since some large VM backup slows down other small VM backup, another optimization in our work is to minimize the average VM backup time by scheduling VM backup in multiple batch rounds. The memory requirement and disk usage for the proposed solution is very small while the overall throughput and backup process timing is not compromised. The tradeoff is that individual backup requests may be delayed while the average overall snapshot backup is optimized. Our evaluation validates this and shows a satisfactory backup throughput in a large cloud cluster setting.

1 Introduction

Periodic archiving of virtual machine (VM) snapshots is important for long-term data retention and fault recovery. 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, 18] that identifies redundant content duplicates among snapshots. One architectural approach is to attach a separate backup system with deduplication support such as ones from EMC [5] or NetAppNetApp to the cloud cluster, and the snaphosts of every virtual machine is periodically transferred to the attached backup system. Such a dedicated backup configuration can be expensive, considering that significant networking and computing resource is required to transfer undeduplicatd data and conduct signature comparison. Our work is targeted applications where data processing and archiving can be conducted in an inexpensive commodity storage cluster using a distributed file system such as the Google file system or Hadoop [6, 12]. Resource sharing is common, especially in the cloud where many computing and storage services are collocated in the same physical machines, so we expect the storage cluster to be non-dedicated.

With resource sharing with other services as an assumption, we are motivated to develop a low-profile backup solution with deduplication.

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 is that deletion of old snapshots compete for computing resource as well, because data dependence created by duplicate relationship among snapshots adds processing complexity.

Among the three factors - time, cost, and deduplication efficiency, our main objective is low cost, so as to not significantly impact primary services. By sacrificing memory a faster deduplication system can be built, but then machines must be dedicated to backup, which goes against the converged architecture model that we are supporting. For daily backup jobs, it is ok if the job takes 2-3 hours, as long as the resource usage is minimal and it can be finished during lower usage hours of the day.

The traditional approach to deduplication 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. The tradeoff is that every machine has to read dirty segments twice and that some deduplication requests are delayed for staged parallel processing. Another tradeoff of our efficient batched approach, related to reading each segment

twice, is that we must maintain a consistent view of the disk between the first and second read. We use Copy on Write (CoW) to acheive this, at the cost of additional disk space. With careful parallelism and buffer management, this multi-stage detection scheme can provide a sufficient throughput for VM backup. To minimize the CoW cost and adjust the balance between total backup time and the backup times for individual VMs, we adopt a multiple-batch approach based on the dual bin packing problem.

2 Background and Related Work

At a cloud cluster node providing access to multiple virtual machines, each instance of a guest operating system runs on a virtual machine (VM), accessing virtual hard disks represented as virtual disk image files in the host operating system. For VM snapshot backups, file-level semantics are normally not provided, and snapshot operations take place at the virtual device driver level. This means no fine-grained file system metadata can be used to determine the changed data.

Further, System admistrators may wish to adjust the balance between resource usage for backup and the time spent performing backups. There may be a small backup window (e.g. 2 hours) to complete the backups each night, but the backup should also minimize impact on VMs running during that time. Other studies have shown, for example, that the cost of Copy on Write (CoW) during daily backups can be as much as 8% of the total data size [11].

Another issue is the sheer size of the backups. to maintain mutiple daily uncompressed snapshots of each VM would be very expensive, even though most of the data remains the same from day to day, and even between VMs common software packages and operating systems mean that the amount of unique data is much smaller. 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 [18] 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 [15, 16]. As discussed in Section 1, there is no dedicated resource for deduplication in our targeted setting and low memory usage is required so that the resource impact to other cloud services is minimized. Approximation techniques which reduce the memory requirement at the expense of deduplication efficiency are studied in [3, 7, 17]. In comparison, this paper focuses on full deduplication without approximation.

Additional inline deduplication techniques are studied in [8, 7, 13]. 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, efficiently finishing the backup of VM images within a reasonable time window is more important than optimizing individual VM block backup requests, and by batch processing the deduplication requests we can decrease resource usage.

3 System Design

We consider deduplication in two levels. The first level uses coarse-grain segment dirty bits for version-based detection [4, 14]. Our experiments with Alibaba's production dataset shows that over 70 percent of duplicates can be detected using segment dirty bits when the segment size is 2M bytes. This setting requires the virtual disk driver to maintain segment dirty bits, which has a negligible space cost (1 bit per 2 MB). In the second level of deduplication, content blocks of dirty segments are compared with the fingerprints of unique blocks from previous snapshots. Our key strategies are explained as follows.

- **Separation of duplicate detection and data backup.** The second level detection requires a global comparison of fingerprints. Our approach is to perform duplicate detection first before actual data backup. This requires a prescanning of dirty VM segments, which does incur an extra round of VM reading. During VM prescanning, detection requests are accumulated. Aggregated deduplicate requests can be processed partition by partition. Since each partition corresponds to a small portion of global index, memory cost to process detection requests within a partition is small and can be handled in memory.
- **Buffered data redistribution in parallel duplicate detection**. Let *global index* be the meta data containing the fingerprint values of unique snapshot blocks in all VMs and the reference pointers to the location of raw data. A logical way to distribute detection requests among machines is based on fingerprint values of content blocks. Initial data blocks follow the VM distribution among machines and the detected duplicate summary should be collected following the same distribution. Therefore, there are four all-to-all data redistribution operations involved. One is to map detection requests from VM-based distribution to fingerprint based distribution, and another one is to map duplicate summary from fingerprint-based distribution to VM based distribution. Two additional

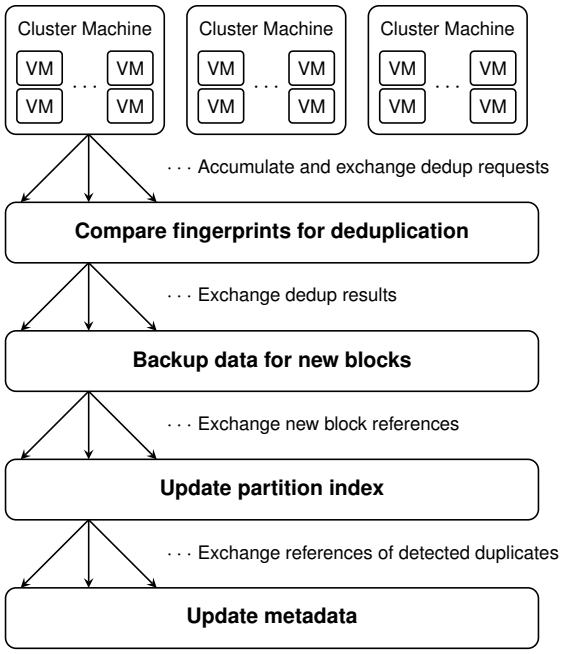

Figure 1: Overall architecture of parallel VM snapshot deduplication

rounds are required to ensure no duplicate blocks are missed and that both the partition index holder and VM owning machines have correct data references. The redistributed data needs to be accumulated on the disk to reduce the use of memory. To minimize the disk seek cost, outgoing or incoming data exchange messages are 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.

- **Round Scheduling**. When all data is scheduled to be deduplicated in one round, the cost of Copy on Write (CoW) can be significant, as explored in other papers[11]. We therefore develop an algorithm to break up the data into multiple rounds to make the best use of the efficiency of batch processing, while minimizing the cost of CoW.

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.A VM image is divided into a set of segments and each segment contains content blocks of variable-size, partitioned using the standard chunking technique 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.

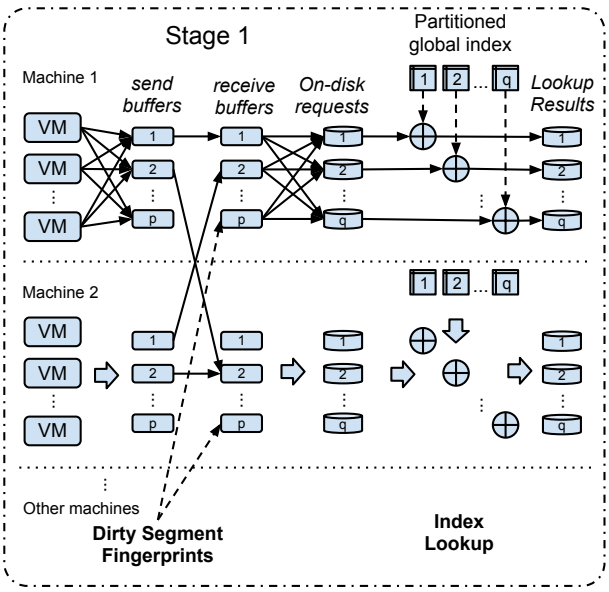

Figure 2: Processing flow of Stage 1.

In Stage 1a, each machine independently reads VM images that need a backup and forms duplicate detection requests. The system divides each dirty segment into a sequence of chunk blocks, computes the meta information such as chunk fingerprints, sends a request to a proper machine, and accumulates received requests into a partition on the local temporary disk storage. The partition mapping uses 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 it 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 parti-

tions. 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, which is depicted in Figure 2, is much more efficient than Option 1 because $2p + q$ is much smaller than $p \times q$, except for the very small values. As a result, each buffer in Option 2 has a bigger size to accumulate requests and that means less disk seek overhead.

Stage 1b 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. One potential issue is that there may be multiple copies of a new block added to the system during the same round. We call these dup-with-new blocks and the redundandy is discovered and logged during the index lookup. Our experience is that there is significant redundancy during the initial snapshot backup and after that, the percentage of redundant blocks due to concurrent processing is small.

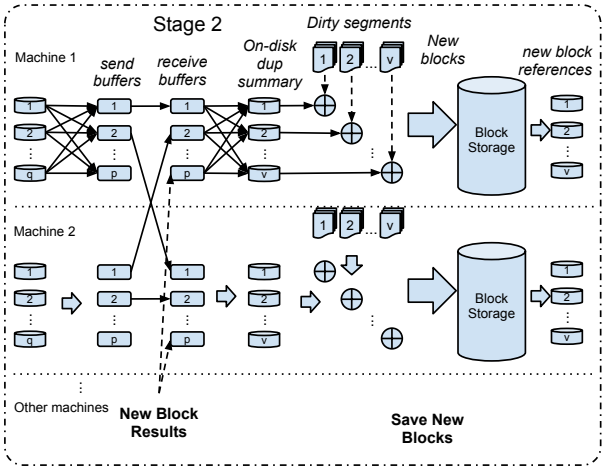

Figure 3: Processing flow of Stage 2.

In stage 2a the deduplication results corresponding to unique (new) data blocks are exchanged using p send and receive buffers to route the responses to the deduplication requesters. Dup-with-new blocks are not exchanged, as they are treated as duplicate blocks, only without data references (the data references to these blocks are added in Stage 3).

Stage 2b is to perform the real backup. 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, references to each new data block are saved so that the global index may be updated.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.

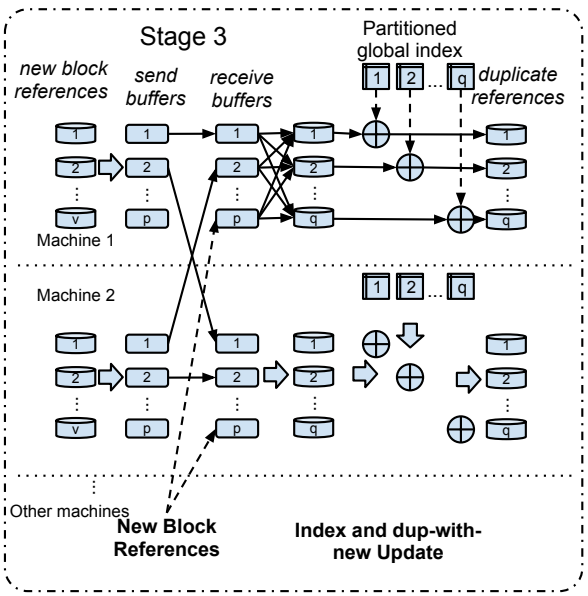

Figure 4: Processing flow of Stage 3.

In Stage 3a the machines exchange references to new blocks. This is required because it is the VM holder that backs up the data, but the partition index holder that must perform index lookups. Each machine reads the references obtained for all new blocks written, and then sends those references to the partition index holder, and the received references are saved to disk during 3a.

Stage 3b is where the index update occurs; new blocks are added to each partition index, and additionally dup-with-new blocks are re-read to obtain references. The dup-with-new blocks, now with references, are added to the duplicate result lists to be returned in Stage 4.

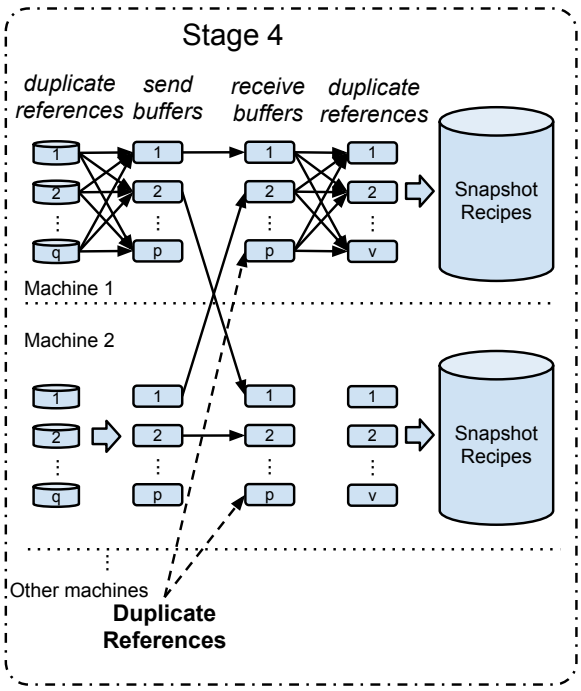

Figure 5: Processing flow of Stage 4.

In the final Stage of our algorithm, Stage 4, All duplicate references, including dup-with-new blocks, are returned to the requesters from the index holders in 4a, And in 4b the snapshot recipes are updated with those pointers.

The above steps can be executed by each machine using one thread to minimize the use of computing resources. The disk storage usage on each machine is fairly small for storing part of the 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. M includes space for network communications and buffering data which is being sent to disk. We have found that as long as the write/read buffers are not unreasonably small (e.g. 4KB), the size of the disk buffers does not have a great impact on backup time, so we allocate small disk buffers (e.g. 128KB read and 2.5MB shared across all write buffers), and then allocate the remaining memory to index space or network buffers, depending on the stage.

- For Stage 1, M is divided for 1) an I/O buffer to read dirty segments; 2) $2p$ send/receive buffers and q disk buffers for deduplication requests.
- 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 blocks to the backup storage; 2) summary of duplicate blocks within dirty segments.

Snapshot deletion. Each VM will keep a limited number of automatically-saved snapshots and expired snapshots are normally deleted. To perform compaction of unused blocks, We adopt the idea of mark-and-sweep [7]. A block or a segment can be deleted if its reference count is zero. To delete unreferenced blocks or segments periodically, we read the meta data of all snapshots and determine which blocks and segments are referenced in parallel. Similar to the multi-stage duplicate detection process, marking is conducted in multiple stages. Stage 1 is to read the segment and block metadata to accumulate references in different machines based on the fingerprint based distribution. Stage 2 is to mark blocks/segments within each partition and detect those records which are unmarked (and therefore can be deleted). The backup data repository logs deletion instructions, and will periodically perform a compaction operation when its deletion log is too big.

4 Multi-round Batch Scheduling

The naïve way for synchronized backup is to schedule all of the backup tasks in one round. Since parallel backup of these tasks goes through several synchronized stages, the backup time of each VM is approximately equal to the total execution of all backup tasks. Given VM sizes are highly skewed in a large cluster (see Fig. 6 for a plot of the distribution in several real clusters), some big VMs will take a long time and other small VM backup tasks synchronized in one round will have to wait. Thus the average backup time for small VMs is fairly large.

There is another disadvantage with a single-round scheduling. During backup period, new data modification requests can lead a inconsistency between data signature read previously and actual data on the disk.

In order to preserve a consistent view of VM images, we employ the Copy-on-Write (CoW) technique to protect the dirty segments against incoming disk writes during the backup period [**?**]. The cost of CoW is a factor of the data size, the write rate, and duration of CoW protection. Previous studies have shown that as much as 8% or even more of total disk capacity may have to be reserved for CoW [11]. Prolonging the backup time of individual VM backup caused by skewed VM size distribution and single-round operation synchronization significantly increases COW overhead.

To determine how evenly distributed VM sizes are, we obtained statistical data for 22 Aliyun clusters. The clusters in our dataset all have a very long tail distribution, though local minima exist in the histogram and the range of VM sizes varies. All of the clusters we have

looked at have similarly shaped distributions. Size distribution histograms for two of the clusters are shown in Fig. 6. Note that there is a long tail on both histrograms which has been cut (out to 100GB for the first, and out to 2000GB for the second). The key insight from these size distributions is that in all of the productions clusters we evaluated, the VM Size distribution is highly skewed, which motivates the importance of round scheduling.

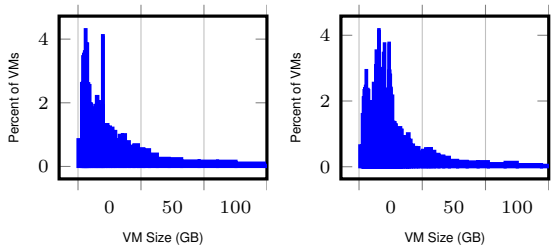

Figure 6: Histograms of average VM sizes for several production clusters

4.1 Cost Functions and Optimization Objectives

When a multi-round schedule is deployed, the backup of a VM can be executed in one of these rounds. Tasks at each round go through the processing stages discussed in Section **??**.

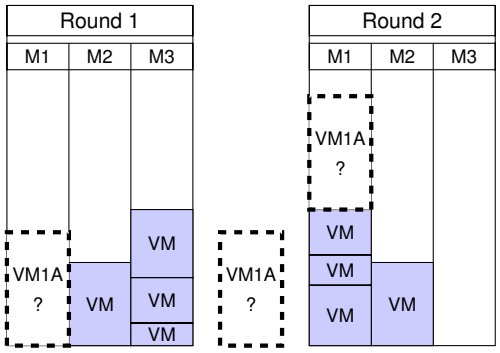

Figure 7: Choosing a round to schedule a VM. New VM marked as VM1A.

The question is which round a VM backup task should be scheduled? Figure 7 illustrates an intermediate step of scheduling in which some of VMs have been assigned to a round, and they are marked with the gray boxes. VM named VM1D is to be scheduled and can be assigned to Round 1 or Round 2. Scheduling VM1D on Round 1 would not significantly increase the execution time of round 1 because machine M1 does not have any VM scheduled.

The disadvantage of having many rounds of backup is that the time for the entire backup job for all VMs is stretched for many iterations. We call this time as the backup job span. In practice, we would like the job span to be controlled within few hours during night time. Thus there is a tradeoff to strike in the design of scheduling and the optimization metrics are listed as follow.

- *Average backup time T*. This is to measure the average number of the backup time time. A smaller number will also shorten the COW cost as discussed above. The average backup time also reflects the average computing resource required. A shorter backup time means that the computer system devotes less resource in the backup of each VM.
- *The job span: J*. This is to measure the total time needed to complete all backup tasks. Longer the job span is, the more burden is imposed to the cluster management. While a short span is better, the constraint is not high as long as the job can be completed in a short period of time (e.g. a few hours during night time).
- *Aggregated optimization*. We will use the following combined cost function to set the optimization objective.

$$\alpha T + (1 - \alpha)J$$

where α varying between 0 and 1 represents the importance of the average backup time.

There are two cost functions involved.

- Function $Load(m, r)$ is the summation of all VMs processed by each physical machine m at each round r.
- The execution of time of each round in completing the back up of the assigned VMs. We can show that the execution time of each round is proportional to the maximum data load of all physical machines.

4.2 Heuristic Algorithms

We assume the physical machine location of each VM is preassigned, and that the VM should be backed up from the machine hosting it. This may not be a requirement of the system (e.g. if the virtual disks are stored on a DFS any machine could theoretically backup the VM), but in a converged storage architecture it should be the case that at least one replica of the VM disk is on/near the hosting machine, so we maximize data locality by taking advantage of this. A scheduling algorithm is developed to decide the number of rounds to be used and the round number that each VM is processed. Our algorithm mapping each VM to a round number is outlined as follows.

- Repeat following steps for all possible k values (numbers of rounds) and choose the schedule with the smallest objective function.

- For given round number k, derive a schedule as follows.
  - Select an unscheduled VM. The default option is to select the VM with the maximum VM size. Another option is to consider the maximum-sized VM in a physical machine which has the highest unscheduled work load.
  - Try all k rounds and find a round number to assign the selected VM. There are two options to select such a round.
    * Select the round which has the smallest value $Load(m, r)$.
    * Select the round that gives the minimum increase of the total execution time.

foreach *k in (1..Num VMs)* **do**
  Sort VMs in descending size order;
  foreach *VM* **do**
    schedule VM to round with smallest load;
  end
  Compute the cost of this schedule (using objective function);
end
Select the k which had the lowest schedule cost;

Algorithm 1: VM round scheduling algorithm

foreach *k in (1..Num VMs)* **do**
  foreach *x in (1..Num VMs* **do**
    Choose the largest VM from the machine with the highest unscheduled workload;
    Schedule VM to round with smallest load;
  end
  Compute the cost of this schedule (using objective function);
end
Select the k which had the lowest schedule cost;

Algorithm 2: Alternate VM round scheduling algorithm

4.3 Properties and Discussions

The round scheduling algorithm, which chooses a round number that has the smallest workload for the corresponding physical machine of a VM, can be competitive to the optimum schedule that minimizes the average backup time. The competitive factor is 2 and we list the analysis as follows.

Property. *The RS2 algorithm delivers a schedule with average backup time T and it satisfies $T \leq 2T_{opt}$.*

5 Evaluation

We have implemented and evaluated a prototype of our multi-stage deduplication scheme on a cluster of dual quad-core Intel Nehalem 2.4GHz E5530 machines with 24GB memory. Our implementation is based on Alibaba's Xen cloud platform [1, 17]. 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.

We have performed a trace-driven study using a 1323 VM dataset collected from a cloud cluster at Alibaba's Aliyun. based on the data distribution from the trace data, we have also simulated the VM size distribution of several much larger clusters by generating synthetic traces (e.g. the two clusters in Fig 6 have 4224 and 8046 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 an average of 40GB of storage data usage on including OS and user data disk (though the exact size of a VM varies with a long-tail distribution).

Each VM image is divided into 2 MB fix-sized segments and each segment is divided into variable-sized content blocks 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 usage of local storage is controlled about 50MB/second for backup in the presence of other I/O jobs. Noted that a typical 1U server can host 6 to 8 hard drives and deliver over 300MB/second. 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.

5.1 Simulated Algorithm Comparison

Based on results from out trace-driven study, we also compare algorithms using the amount of data transferred over the network, written to disk, and read from disk in each stage. From this simple performance model, we can compare different scheduling heuristics and weights and choose the most appropriate algorithm for a set of requirements. Our simulation uses VM size distributions from production clusters to evaluate the scheduling algorithms across several clusters. we discuss the size dis-

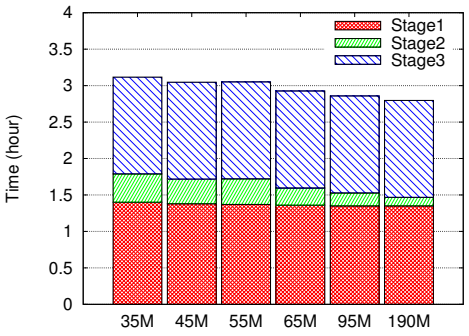

Figure 8: Parallel time when memory limit varies.

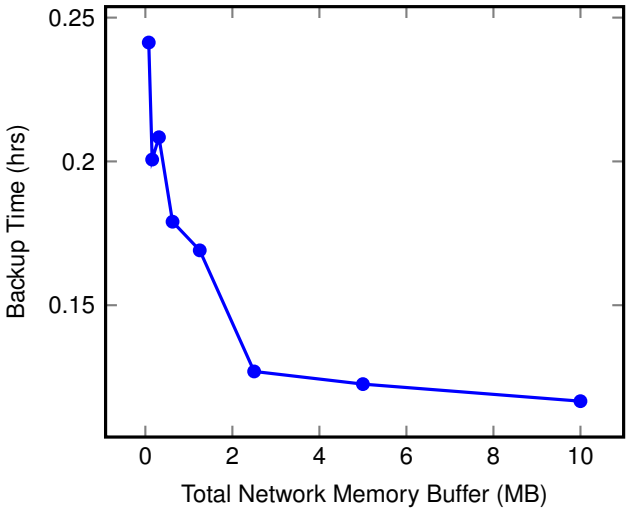

Figure 9: Backup time for verying amounts of memory allocated to network communication. Other Settings: 10 Machines, 5x40GB VMs per machine, write buffer 6.25MB, Read Buffer 128KB

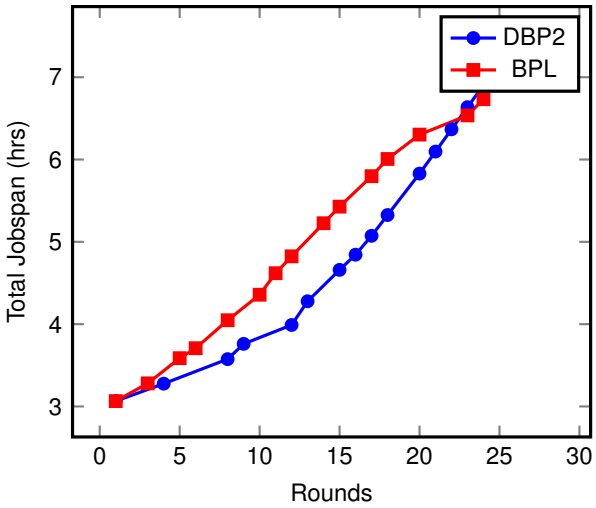

Figure 10: Jobspan for DBP2 and BPL

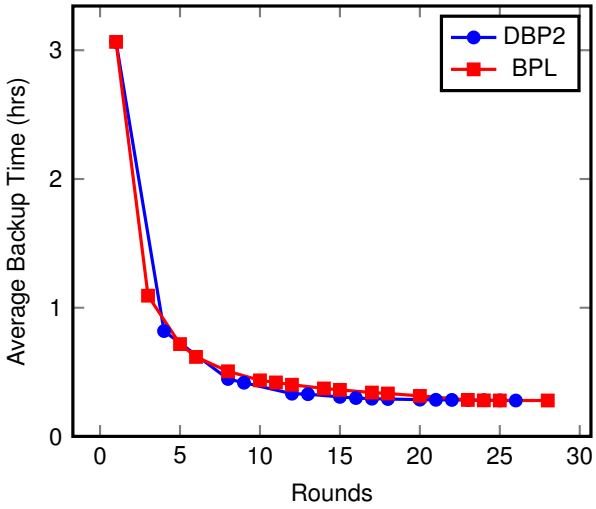

Figure 11: Average Backup Time for DBP2 and BPL

tributions in section **??**.

The names of the algorithms are just the temporary names I've been using for coding, DBP2 is the relative of the dual bin packing algorithm, BPL is the machine by machine largest VM to shortest round algorithm

Figure 8 shows the total parallel time in hours to backup 2500 VMs on a 100-node cluster a when limit M imposed on each node varies. 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 segments, and final data copying to the backup storage is overlapped with VM scanning. During dirty 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 190MB. The aggregated deduplication throughput is about 8.76GB 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 3.1 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 limit M=35MB is imposed and the number of partitions per machine (q) varies. Row 2 is memory space required to load a partition of global index and detection requests. When $q = 100$, the required memory is 83.6 MB and this exceeds the limit M =35MB. Row 3 is the parallel time and Row 4 is the aggregated throughput of 100 nodes. Row 5 is the parallel time for using Option 1 with

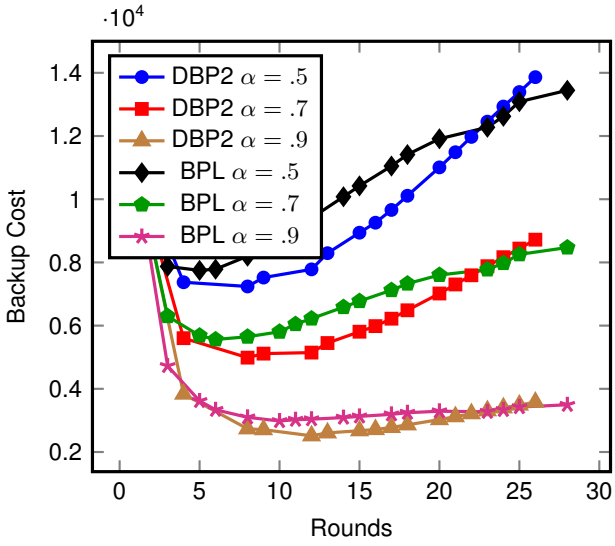

Figure 12: Backup Cost for DBP2 and BPL. TODO:reformat this graph

$p \times q$ send buffers described in Section 3. When q increases, the available space per buffer reduces and there is a big increase of seek cost. The main network usage before performing the final data write is for request accumulation and summary output. It lasts about 20 minutes and each machine exchanges about 8MB of metadata per second with others during that period, which is 6.25% of the network bandwidth.

Table 1: Performance when M=35MB and q varies.

#Partitions (q)	100	250	500	750	1000
Index+request (MB)	83.6	33.5	16.8	11.2	8.45
Total Time (Hours)	N/A	3.12	3.15	3.22	3.29
Throughput GB/s	N/A	8.76	8.67	8.48	8.30
Total time (Option 1)	N/A	7.8	11.7	14.8	26

6 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 proposed scheme is resource-friendly to the existing cloud services. The evaluation shows that the overall deduplication time and throughput of 100 machines are satisfactory with about 8.76GB 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. Our future work is to conduct more experiments with production workloads.

Acknowledgment. We thank Hong Tong for his help and support. This work is supported in part by NSF 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 National Science Foundation.

References

[1] Alibaba Aliyun. 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.

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

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

[6] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *SOSP*, pages 29–43. ACM, 2003.

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

[8] 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.

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

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

[11] H. Shim, P. Shilane, and W. Hsu. Characterization of incremental data changes for efficient data protection. In *ATC'13*, pages 157–168. USENIX.

[12] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *MSST'10*, pages 1–10.

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

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

[15] J. Wei, H. Jiang, K. Zhou, and D. Feng. MAD2: A scalable high-throughput exact deduplication approach for network backup services. In *MSST'10*, pages 1–14. IEEE, May 2010.

[16] 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.

[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] 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.