VM-Centric Snapshot Deduplication for Cloud Data Backup

Wei Zhang*†, Daniel Agun*, Tao Yang*, Rich Wolski*, Hong Tang‡
*University of California at Santa Barbara
†Pure Storage Inc.
‡Alibaba Inc.
Email: wei@purestorage.com, {dagun, tyang, rich}@cs.ucsb.edu, hongtang@alibaba-inc.com

***Abstract*—Data deduplication is important for snapshot backup of virtual machines (VMs) because of excessive redundant content. Fingerprint search for source-side duplicate detection is resource intensive when the backup service for VMs is co-located with other cloud services. This paper presents the design and analysis of a fast VM-centric backup service with a tradeoff for a competitive deduplication efficiency while using small computing resources, suitable for running on a converged cloud architecture that cohosts many other services. The design consideration includes VM-centric file system block management for the increased VM snapshot availability. This paper describes an evaluation of this VM-centric scheme to assess its deduplication efficiency, resource usage, and fault tolerance.**

I. Introduction

Commercial "Infrastructure as a Service" clouds (i.e. *public clouds*) often make use of commodity data center components to achieve the best possible economies of scale. In particular, large-scale e-commerce cloud providers such as Google and Alibaba deploy "converged" components that co-locate computing and storage in each hardware module (as opposed to having separate computing and storage "tiers.") The advantage of such an approach is that all infrastructure components are used to support paying customers – there are no resources specifically dedicated to cloud services. In particular, these providers use software to aggregate multiple direct attached low-cost disks together across servers as a way of avoiding the relatively high cost of network attached storage [12], [24], [19]. In such an environment, each physical machine runs a number of virtual machines (VMs) and their virtual disks are stored as disk image files. Frequent snapshot backup of virtual machine images can increase the service reliability by allowing VMs to restart from their latest snapshot in the event of a server failure. Snapshots contain highly redundant data chunks and deduplication of redundant content [21], [32] is necessary to substantially reduce the storage demand.

Source-side deduplication [28], [27], [11], [30] eliminates duplicates before backup data is transmitted to a secondary storage, which saves network bandwidth significantly; however its resource usage can impact other co-located cloud services. It is memory-intensive to compare a large number of fingerprints and identify duplicates, even with optimization or approximation techniques developed [13], [8], [3], [10]. Another side effect of deduplication is the possible loss of failure resilience [4]. If a shared block is lost, all files that share that block are affected. A cloud may offload backup workload from production server hosts to dedicated backup proxy servers (e.g. EMC Data Domain) or backup services (e.g. Amazon S3). This approach simplifies the cluster architecture and avoids potential performance degradation to production applications when backups are in progress. However, sending out raw and undeduplicated backup data wastes a huge amount of network bandwidth that would otherwise be available to user VMs.

Our work considers a backup service with source-side deduplication for converged cloud architectures that is co-located with user VMs, sharing the cloud compute, network, and storage resource with them. This paper extends our preliminary simulation study [31] to restrict the scope of cross-VM deduplication and simplify the deduplication process by separating popular data chunks and presents the design and implementation of a VM-centric backup scheme. We term this approach *VM-centric* because the deduplication algorithm considers VM boundaries in making its decisions as opposed to treating all blocks as being equivalent within the storage system. Our work focuses on a tradeoff that allows the sharing of only "popular" data blocks across virtual machines while using localized deduplication within each VM to achieve both storage savings and fault isolation. Another issue addressed is fault isolation in a content sharing environment caused by deduplication. One consequence of aggressive deduplication is the possible loss of failure resilience. If a shared block is lost, all files that share that block are affected [4] and our work is motivated by this to develop VM-centric fault isolation techniques.

The main contribution of this paper is 1) a design and implementation of a backup service to integrate the VM-centric techniques while minimizing the resource impact on other collocated cloud services; 2) a VM-centric file block management to improve snapshot availability by separating popular chunk replication and packaging data chunks from the same VM into a file system block as much as possible; 3) an analysis on the deduplication efficiency by using popular items and also on snapshot availability improved by the above fault isolation design. The analysis provides insights and justification on the proposed VM-centric strategies. The integrated optimization includes similarity-guided local deduplication for

978-1-4673-7619-8/15/$31.00

each VM to make the overall deduplication efficiency more competitive to a traditional deduplication scheme. Our experimental study compares the VM-centric approach with the previous VM-oblivious approaches in terms of deduplication efficiency, resource usage, and snapshot availability. Compared to our earlier study [31] which achieves 92% of what the full deduplication can do, our integrated system delivers over 96% in a tested dataset from Alibaba cloud.

The rest of this paper is organized as follows. Section II reviews the background and discusses the design options for snapshot backup with a VM-centric approach. Section III presents our scheme for snapshot deduplication. Section IV describes a system implementation that evaluates the proposed techniques. Section V is our experimental evaluation that compares with other approaches. Section VI concludes this paper. The appendix describes an analysis on the deduplication efficiency with top k popular chunks.

II. Background and Design Considerations

Figure 1 illustrates a converged IaaS cloud architecture where each commodity server hosts a number of virtual machines and storage of these servers is clustered using a distributed file system [12], [24]. Each physical machine hosts multiple virtual machines. Every virtual machine runs its own guest operating system and accesses virtual hard disks stored as image files maintained by the operating system running on the physical host. For VM snapshot backup, file-level semantics are normally not provided. Snapshot operations take place at the virtual device driver level, which means no fine-grained file system metadata can be used to determine the changed data. In a converged setting with source-side deduplication, the resources that are used to implement snapshot backup and deduplication are the same resources that must support cloud-hosted VMs. Thus the backup service collocated with other cloud services has to minimize its resource impact.

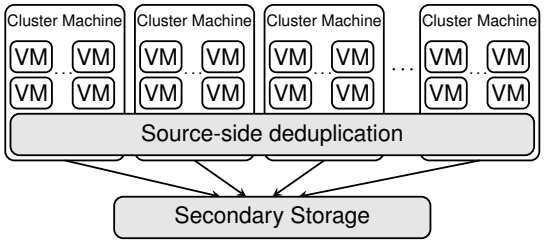


Fig. 1. VM snapshot backup running on a converged cloud cluster.

File backup systems have been developed to use fingerprints generated for data "chunks" to identify duplicate content [21], [23]. Source-side deduplication is studied at a chunk or file level [28], [27] for client data backup and can be integrated with global deduplication among multiple clients [11]. In this context, the client side is the source-side and the client-side computing resource restriction is not a primary concern. Our work considers the source-side deduplication in a cloud cluster environment before data is transferred to a backup storage and the computing resource constraint at the cluster side is a major concern.

It is expensive to compare a large number of fingerprints so a number of techniques have been proposed to improve duplicate identification. For example, the Data Domain method [32] uses an in-memory Bloom filter to identify non-duplicates and a prefetching cache for potential duplicates that might hit in the future. Additional inline deduplication and approximation techniques are studied in the previous work [3], [17], [26], [31].

Even with these optimization and approximation techniques, resource usage such as memory for deduplication is still extensive for a shared cloud cluster. For example, in the experiments discussed in Section V-A, the raw snapshot data has a size of 1,000TB on 100 physical machines that host VMs, cross-machine fingerprint comparison using Bloom filter and approximated routing [32], [8] still needs several gigabytes of memory per machine. This can impact other primary cloud services sharing the same computing resource. Our objective is to have an approximation scheme which uses no more than a few hundred megabytes of memory during normal operation. Thus the desired ratio of raw data to memory ratio is from 100K:1 to 30K:1. Our proposed scheme achieves 85K:1 and this ratio is about the same as the number of machines increases.

Index sampling with a prefetch cache [13] is proposed for efficient single-machine deduplication. This scheme uses 25GB memory per 500TB of raw data and thus the raw data to memory ratio is 20K:1. The scheme proposed in this paper uses three times less memory. We have not incorporated this index sampling technique because it is difficult to extend for a distributed architecture. To use a distributed memory version of the sampled index, every deduplication request may access a remote machine for index lookup and the overall overhead of access latency for all requests is significant.

While deduplication reduces storage cost, it creates an artificial data dependence among VMs: when a shared data chunk is lost in the storage, all VMs that refer such chunks after deduplication face a data loss. Such an issue has been identified in file systems [4] and more replication is used for shared data. we follow this idea and study VM-centric strategies. When the backup storage is implemented using a distributed file system such as Hadoop and Google file system [12], the file system block size is often chosen to be large (e.g. 64MB). On the other hand, deduplication implementations [13], [3], [32], [14], [8] typically use smaller chunk sizes (e.g. 4K bytes). Thus we need to address the size gap between file system blocks and chunks.

III. VM-centric Snapshot Deduplication

With the considerations discussed in the previous section, we propose a VM-centric approach (called VC) for a co-located backup service that has a resource usage profile suitable for use with converged cloud architectures. This compares the traditional deduplication approach *VM-oblivious* (VO) which manages duplicate data chunks without consideration

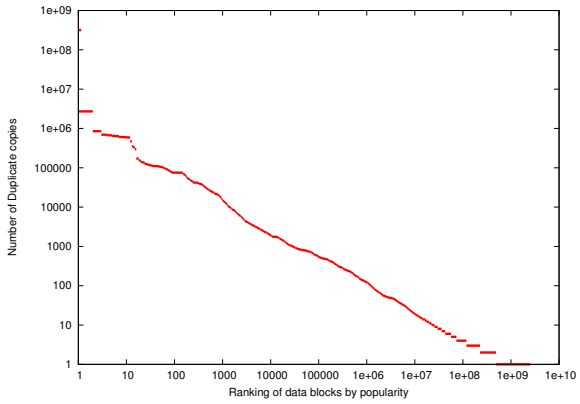


Fig. 2. Duplicate frequency versus chunk ranking in a log scale after local deduplication.

of VMs. Section III-A discusses our key ideas and design for duplicate detection.

A. VM centric strategies

We separate the deduplication within a VM and cross VMs and simplify the cross-VM deduplication while maximizing the inter-VM deduplication efficiency as much as possible. This is because global deduplication that detects the appearance of a chunk in any VM requires a substantial resource for fingerprint comparison. To simplify cross-VM deduplication, we restrict the search scope of deduplication within the top k most popular items. This popular data set is called the **PDS**.

What motivates us to use PDS is that for those chunks that appear in different VMs, the top k most popular items dominate the appearance distribution following our previous evaluation [31]. Figure 2 shows the distribution of chunk popularity for a data trace from Alibaba with 2500 VMs discussed in Section V. We define chunk popularity as the number of unique copies of the chunk in the data-store, i.e., the number of copies of the chunk after local deduplication. The distribution from this figure is Zipf-like. Let σ denote as the percentage of unique data chunks belonging to PDS and from the evaluation in Section V, we find that σ with about 2% can deliver a fairly competitive deduplication efficiency.

PDS can be computed in an offline manner periodically, e.g., on a monthly basis. Compared to [32], the fingerprint index size is reduced by $100-\sigma$ percentage (e.g. 98% with $\sigma = 2\%$) and the searchable fingerprint space becomes very small under the popularity constraint. The fingerprint-guided distributed mapping in [3], [8] narrows the search scope of each data chunk, but it does not reduce th total amount of searchable fingerprints used for the entire deduplication. Appendix A provides an analysis of the k value selection and PDS effectiveness.

We describe two complementary strategies below for the above VM-centric design.

- **VM-specific duplicate search optimization** – While cross-VM deduplication is simplified, we intend to optimize the VM-specific deduplication under a reasonable memory consumption to make up the loss of deduplication opportunities due to the cross-VM popularity constraint. We start with the standard version-based detection [7], [28] to identify changed content with dirty bits in a coarse grain segment level. A segment is essentially a super-chunk containing many data chunks. The reason to choose a coarse grain segment level is that since every write for a segment will touch a dirty bit, the device driver maintains dirty bits in memory and cannot afford a small segment size. It should be noted that dirty bit tracking is supported or can be easily implemented in major virtualization solution vendors.

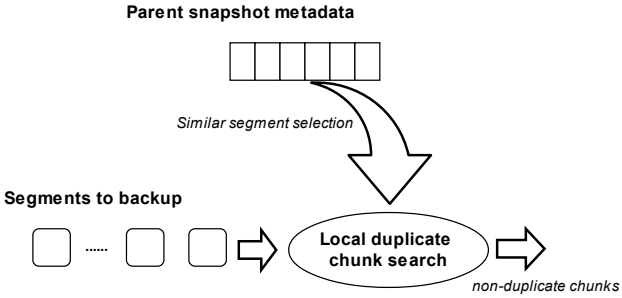


Fig. 3. Similarity-guided local duplicate detection

Since the previous work typically uses a non-uniform chunk size with an average of 4KB or 8KB for the best deduplication effectiveness [13], [3], [32], [8], we conduct additional local similarity guided deduplication by comparing chunk fingerprints of a dirty segment in a snapshot with those in *potentially similar* segments from its parent snapshot. Similarity-guided search can be desirable for cases that data chunks are duplicated irregularly. For example, we found from the Alibaba's cloud cluster dataset that data segment movement happens frequently and files are often rewritten rather than being modified in place. A dirty-bit or offset-based detection is not able to detect such a movement.

We consider a parent segment is potentially similar to the current dirty segment if 1) the parent segment is at the same offset as the current segment. 2) the signature of these segments is the same. The signature of a segment is defined as the minimum value of all its chunk fingerprints computed during backup and is recorded in the snapshot metadata (called recipe). This above approach is motivated by the previous work for similarity-based deduplication [6], [3], [8], [2] while our focus is local search.

When processing a dirty segment, its similar segments can be found easily from the parent snapshot metadata. Then metadata of the similar segments is loaded to memory, containing chunk fingerprints to be compared. To control the time cost of search, we set a limit on the number of similar segment recipes to be fetched. For example, assume that a segment is of size 2MB, its segment recipe is roughly 19KB which contains about

500 chunk fingerprints and other chunk metadata. By limiting at most 10 similar segments to search, the amount of memory for maintaining those similar segment recipes is 190K, which is insignificant.

- **VM-centric file system block management** – When a chunk is not detected as a duplicate to any existing chunk, this chunk will be written into a file system block. In addition to the fact that a distributed file system block [12], [24] is often configured to be much larger than a chunk, a number of chunks can be combined together and compressed further using a standard compression method such as LZO. Thus a backend file system block contains a large number of compressed chunks. We set the following two constraints in composing chunks for a file system block: 1) Each file system block is either dedicated to non-PDS chunks, or to PDS-chunks. 2) A non-PDS file system block is only associated with one VM.

  Restricting the association with one VM improves fault isolation when some file blocks are lost during storage failures. In addition, storing PDS chunks separately from non-PDS chunks allows special replication handling for those popular shared data. If we do not separate the popular chunks from the less-popular, the popular chunks are dispersed across all of the filesystem blocks in the storage system and we would have to add extra replications for *all* file blocks in order to follow the popularity-driven replication idea from [4]. That reduces the storage efficiency.

The underlying distributed file system has a fixed replication degree for all file system blocks [12], [24]. We add extra replicas for file blocks containing PDS chunks and this brings additional failure protection for these chunks shared by many VMs.

B. Fault Tolerance and Snapshot Availability

We analyze the impact of losing d physical machines to the VM centric and oblivious approaches. There are two impacts to VC. 1) Some PDS fingerprint lookup services do not respond. As a result, some duplicates are not detected and deduplication efficiency suffers, but the overall systems can still function well and fault tolerance is not affected. As discussed in Section IV, PDS index is distributed among all machines in our implementation and thus a percentage of failed nodes causes an increase in missed duplicates proportionally. 2) Some storage nodes do not respond and file blocks on those machines are lost. The availability of VM snapshots is affected and we analyze this impact as follows.

To compute the full availability of all snapshots of a VM, we estimate the number of file system blocks per VM and the probability of losing a snapshot file system block of a VM in each approach as follows. Parameters used in our analysis below are defined in Table I.

As illustrated in Figure 4, we build a bipartite graph representing the association from unique file system blocks to their corresponding VMs in two approaches. For VC, each VM has an average number N_1 of non-PDS file system blocks and has an average of N_2 PDS file system blocks. Each non-PDS block is associated with only one VM. By counting outgoing edges from VMs in Figure 4(a), we get:

$$n * N_1 = \text{ Number of non-PDS file system blocks in VC.}$$

For VO, by counting outgoing edges from VMs in Figure 4(b) with parameters defined in Table I, we have

$$n * N_o = V_o * \text{Number of file system blocks in VO.}$$

Since we choose 2-4% of unique chunks for PDS and Section V-A shows that the deduplication efficiency of VC is very close to that of VO, the number of non-PDS file blocks in VC is fairly close to the number of file blocks in VO. Then

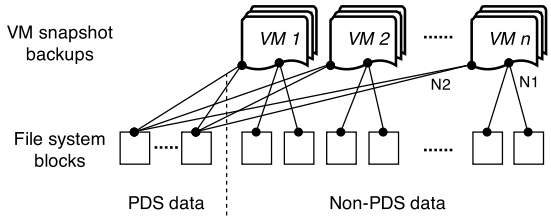
$$\frac{N_o}{N_1} \approx V_o.$$

Figure 5 shows N_1, N_2, and N_o values of 105 VMs from a test dataset discussed in Section V when increasing the number of VMs. N_1 is much smaller than N_o as the formula shows above.

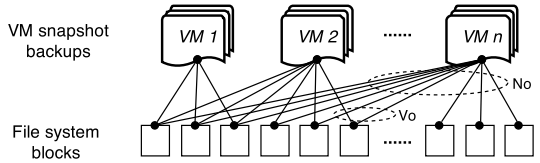
Given d failed machines and r replicas for each file block, the availability of a file block is the probability that all of its replicas do not appear in any group of d failed machines

r, r_c	Replication degree of non-PDS and PDS file blocks in VC. r is also replication degree in VO.
n, p	Number of virtual and physical machines in the cluster
N_1, N_2	The average no. of non-PDS and PDS file blocks in a VM in VC
N_o, V_o	The average no. of file blocks in a VM and the average no. of VMs shared by a file system block in VO
A(r)	Availability of a file block with r replicas and d failed physical machines

TABLE I
MODELING PARAMETERS



(a) Sharing of file system blocks under VC



(b) Sharing of file system blocks under VO

Fig. 4. Bipartite association of VMs and file blocks.

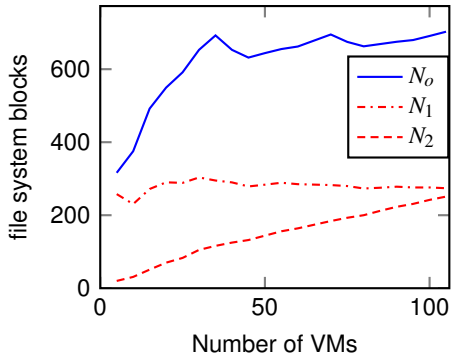


Fig. 5. Measured average number of 64MB file system blocks used by a single VM in VC and VO.

Failures (d)	$A(r_c) \times 100\%$		
	$r_c = 3$	$r_c = 6$	$r_c = 9$
3	99.999381571	100	100
5	99.993815708	100	100
10	99.925788497	99.999982383	99.999999999
20	99.294990724	99.996748465	99.99999117

TABLE II
$A(r_c)$ VALUES IN A CLUSTER WITH P=100 NODES.

among p nodes [9]. Namely,

$$A(r) = 1 - \binom{d}{r} / \binom{p}{r}.$$

Then the availability of one VM's snapshot data under VO approach is the probability that all its file blocks are unaffected during the system failure:

$$A(r)^{N_o}.$$

For VC, there are two cases: $r \leq d < r_c$ and $r_c \leq d$.

- $r \leq d < r_c$: In this case there is no PDS data loss and we need to look at the non-PDS data loss. The full snapshot availability of a VM is:

  $$A(r)^{N_1}.$$

  Since N_1 is typically much smaller than N_o, the VC approach has a higher availability of VM snapshots than VO in this case.
- $r_c \leq d$: Both non-PDS and PDS file system blocks in VC can have a loss. The full snapshot availability of a VM in the VC approach is

  $$A(r)^{N_1} * A(r_c)^{N_2}.$$

  That is still smaller than that of V_O based on the observations of our data. There are two reasons for this: 1) N_1 is much smaller than N_o and we are observing that $N_1 + N_2 < N_o$. 2) $A(r) < A(r_c)$ because $r < r_c$.

Table II lists the $A(r)$ values with different replication degrees, to demonstrate the gap between $A(r)$ and $A(r_c)$.

IV. System Design and Implementation

We describe the design and implementation of a VM-centric backup system that runs on a cluster of Linux machines with Xen-based VMs and the QFS [20] distributed file system. All data needed for the backup service including snapshot data and metadata resides in this distributed file system. One physical node hosts tens of VMs, each of which accesses its virtual machine disk image through the virtual block device driver (called TapDisk[29] in Xen).

A. Per Node Software Components

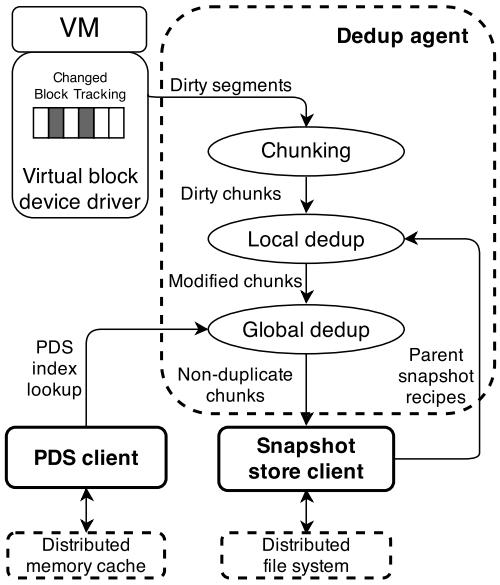


Fig. 6. Data flow during snapshot backup

As depicted in Figure 6, there are four key service components running on each cluster node for supporting backup and deduplication: 1) a virtual block device driver, 2) a snapshot deduplication agent, 3) a snapshot store client to store and access snapshot data, and 4) a PDS client to support PDS metadata access.

We use the virtual device driver in Xen that employs a bitmap to track the changes that have been made to the virtual disk. Every bit in the bitmap represents a fixed-sized (2MB) segment, indicating whether the segment has been modified since last backup. Segments as super-chunks are further divided into chunks using a content-based chunking algorithm [15], which brings the opportunity of fine-grained deduplication. When the VM issues a disk write, the dirty bit for the corresponding segment is set and this indicates such a segments needs to be checked during snapshot backup. After the snapshot backup is finished, the driver resets the dirty bit map to a clean state. For data modification during backup, copy-on-write protection is set so that backup can continue to copy a specific version while new changes are recorded.

The representation of each snapshot has a two-level index data structure. The snapshot meta data (called snapshot recipe) contains a list of segments, each of which contains segment

metadata of its chunks (called segment recipe). In snapshot and segment recipes, the data structures include reference IDs to the actual data location to eliminate the need for additional indirection.

B. A VM-centric Snapshot Store

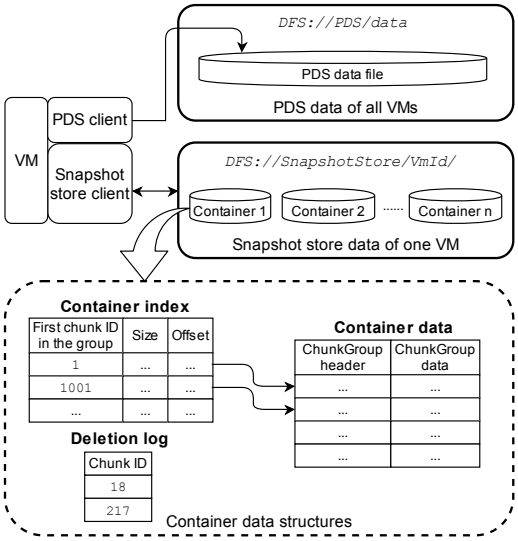


Fig. 7. VM snapshot store data structures

We use the QFS distributed file system to hold snapshot backups. Following the VM-centric idea for the purpose of fault isolation, each VM has its own snapshot store, containing new data chunks which are considered to be non-duplicates. As shown in Figure 7, we explain the data structure of the snapshot stores as follows. There is an independent store containing all PDS chunks shared among different VMs as a single file. Each reference ID to a PDS data chunk in the PDS index is the offset within the PDS file. Additional compression is not applied because for the data sets we have tested, we only observed limited spatial locality among popular data chunks. On average the number of consecutive PDS index hits is lower than 7, thus it is not very effective to group a large number of chunks as a compression and data fetch unit. For the same reason, we decide not to take the sampled index approach [13] for detecting duplicates from PDS as limited spatial locality is not sufficient to enable effective prefetching for sampled indexing. It should be noted that the above spatial locality number for PDS data is subject to application characteristics and we plan to study further on this issue in the future.

PDS data are re-calculated periodically in an offline manner, but the total data size is small. When a new PDS data set is computed, the in-memory PDS index is replaced, but the PDS file on the disk appends the new PDS data identified and the growth of this file is very slow. The old data are not removed because they can still be referenced by the existing snapshots. A periodic cleanup is conducted to remove unused PDS chunks (e.g. every few months).

For non PDS data, the snapshot store of a VM is divided into a set of containers and each container is approximately 1GB. The reason for this division is to simplify the compaction process conducted periodically. When chunks are deleted from old snapshots, chunks without any reference from other snapshots can be removed by this compaction process. By limiting the size of a container, we can control the length of each round of compaction. The compaction can work on one container at a time and move the in-use data chunks to another container. The 1GB setting is based on our experience that compaction performs well and can complete in a reasonable amount time.

Each non-PDS container is further divided into a set of chunk groups for compression. Each chunk compression group is composed of a set of data chunks and is the basic unit for compression and for data access and retrieval. In writing a chunk during backup, the system accumulates data chunks and stores the entire group as a unit after compression. This compression can reduce data by several times in our tested data. For the experiments, we use LZO2 which is a lossless compression library and does not affect snapshot data availability. When accessing a particular chunk, its chunk compression group is retrieved from the storage and decompressed. Given the high spatial locality and usefulness of prefetching in snapshot chunk accessing [13], [23], retrieval of a data chunk group naturally works well with prefetching. A typical chunk compression group contains 1000 chunks in our experiment and this setting is effective for the VM snapshot dataset experienced at Alibaba. Each non-PDS data container is represented by three files: 1) the container data file holds the actual content, 2) the container index file is responsible for translating a data reference into its location within a container, and 3) a chunk deletion log file records all the deletion requests within the container.

A non-PDS data chunk reference ID stored in the index of snapshot recipes is composed of two parts: a container ID with 2 bytes and a local chunk ID with 6 bytes. Each container maintains a local chunk counter and assigns the current number as a chunk ID when a new chunk is added to this container. Since data chunks are always appended to a snapshot store during backup, local chunk IDs are monotonically increasing. When a chunk is to be accessed, the segment recipe contains a reference ID pointing to a data chunk, which is used to lookup up the chunk data as described shortly.

Three API calls are supported for data backup:

- **Append()**. For PDS data, the chunk is appended to the end of the PDS file and the offset is returned as the reference. This operation may only be used during PDS recalculation. For non-PDS data, this call places a chunk into the snapshot store and returns a reference ID to be stored in the recipe of a snapshot. The write requests to append chunks to a VM store are accumulated at the client side. When the number of write requests reaches a fixed group size, the client compresses the accumulated chunk group, adds a chunk group index to the beginning of the group, and then appends the header and data to the corresponding VM file. A new container index entry is also created for each chunk group and is written to the

corresponding container index file.

- **Get()**. Fetching PDS data is straightforward since each reference ID contains the file offset, and the size of a PDS chunk is available from a segment recipe. We also maintain a small data cache for the PDS data service to speedup common data fetching. To read a non-PDS chunk using its reference ID with container ID and local chunk ID, the snapshot store client first loads the corresponding VM's container index file specified by the container ID, then searches the chunk groups using their chunk ID coverage. After that, it reads the identified chunk group from DFS, decompresses it, and seeks to the exact chunk data specified by the chunk ID. Finally, the client updates its internal chunk cache with the newly loaded content to anticipate future sequential reads.
- **Delete()**. Chunk deletion occurs when a snapshot expires or gets deleted explicitly by a user. When deletion requests are issued for a specific container, those requests are simply recorded into the container's deletion log initially and thus a lazy deletion strategy is exercised. Once local chunk IDs appear in the deletion log, they will not be referenced by any future snapshot and can be safely deleted when needed. This is ensured because we only dedup against the direct parent of a snapshot, so the deleted snapshot's blocks will only be used if they also exist in other snapshots. Periodically, the snapshot store identifies those containers with an excessive number of deletion requests to compact and reclaim the corresponding disk space. During compaction, the snapshot store creates a new container (with the same container ID) to replace the existing one. This is done by sequentially scanning the old container, copying all the chunks that are not found in the deletion log to the new container, and creating new chunk groups and indices. Every local chunk ID however is directly copied rather than re-generated. This process leaves holes in the chunk ID values, but preserves the order and IDs of chunks. As a result, all data reference IDs stored in recipes are permanent and stable, and the data reading process is as efficient as before. Maintaining the stability of chunk IDs also ensures that recipes do not depend directly on physical storage locations, which simplifies data migration.
  It should be emphasized that the deleted chunks are marked in the log, but they do not get deleted until compaction time. Only compaction process will read and use this information. New snapshots should not refer a chunk marked as deletion and referenced data will never enter the delete log.

V. Evaluation

We have implemented and evaluated a prototype of our VC scheme on a Linux cluster of machines with 8-core 3.1Ghz AMD FX-8120 and 16 GB RAM. Our implementation is based on the Alibaba cloud platform [1], [31] and the underlying DFS uses QFS with default replication degree 3 while the PDS replication degree is 6. Our evaluation objective is to study deduplication efficiency and resource usage of VC, and assess its processing time and backup throughput, and the benefit in fault tolerance. We will compare VC with a VO approach using stateless routing with binning (SRB) based on the previous work [8], [3]. SRB executes distributed deduplication by routing data chunks to cluster machines [8] using a min-hash function [3]. Once a data chunk is routed to a machine, the chunk is compared with the fingerprint index within this machine locally [3]. While this VO approach is designed for general deduplication and data backup, we choose it as a baseline for comparison because the other approaches would have a similar level of computing resource usage.

We have performed a trace-driven study based on a production dataset from Alibaba Aliyun's cloud platform [1] with about 2500 VMs, running on 100 physical machines. Each machine in the production environment is more powerful than our evaluation cluster and has 16 cores with 48GB memory. Each machine hosts up to 25 VMs and each VM keeps 10 automatically-generated snapshots in the storage system while a user may instruct extra snapshots to be saved. Each VM has about 40GB of storage data on average including OS and user data disk while the size distribution is highly skewed and the maximum to the average ratio is close to 40. Each physical machine deals with about 10TB of snapshot data and each 1TB data represents one snapshot version of 25 VMs. Thus the total amount of raw snapshot in 100 physical machines is about 1,000TB. The VMs of the sampled data set use popular operating systems such as Debian, Ubuntu, Redhat, CentOS, win2008 and win2003. Each machine hosts a variety of applications running these operating systems including web services and database management. The amount of daily data update is not significant and the daily snapshot change rate is about 1-3% on average. The fingerprint for variable-sized chunks is computed using their SHA-1 hash [18], [22]. Note that variable sizing is not required in our scheme.

A. Deduplication Efficiency and Memory Usage

	VC			VO
	No PDS no simil	No PDS simil	2%PDS simil	SRB
Dedup Efficiency	75.86%	87.91%	96.01%	97.86%
Memory per-machine	N/A	10MB	118MB	2.4GB

TABLE III
DEDUPLICATION EFFICIENCY AND PER-MACHINE MEMORY USAGE FOR DIFFERENT VC SETTINGS AND SRB.

Table III shows the deduplication efficiency and per-machine memory usage for SRB and VC with different settings. Deduplication efficiency is defined as the percent of duplicate chunks removed, comparing to a perfect scheme which detects and removes all duplicates. Notice σ is the percentage of unique chunks selected in PDS. With $\sigma = 2\%$, Column 4 shows its deduplication efficiency can reach over 96%. The loss of efficiency in VC is caused by the restriction

of the physical memory available in the cluster for fast in-memory PDS index lookup. Memory usage per machine is low because each machine only hosts $1/p$ of index for PDS plus some buffer space where p is the number of physical machines. SRB in Column 5 can deliver up to 97.86% deduplication efficiency, which is slightly better than VC. Thus this represents a tradeoff as VC uses much less resource and faster deletion. Memory usage per machine in SRB includes the Bloom filter space to access the on-disk index and cache for frequently or recently accessed chunk index.

Column 2 of Table III shows deduplication efficiency of VC without using PDS and local similarity search. It just relies on the file segment dirty bits maintained in the Alibaba virtual machine platform. Thus there is no memory cost (marked as N/A). Thus segment-level version detection reduces the data size to about 24.14% of original data, which is a 75.86% reduction. Namely 10TB snapshot data per machine is reduced to 2.414TB. Column 3 shows the deduplication result with similarity-guided local. It can further reduce the data size to about 1.205T, which is 12.05% of original. Thus it delivers a 50.08% reduction after the version-based deduplication. The popularity-guided global deduplication with $\sigma = 2\%$ reduces the data further to 860GB, namely 8.6% of the original size. So it provides additional 28.63% reduction. The overall memory usage of VC for each physical machine is very small, which is insignificant to other hosted cloud services. In comparison, the 2.4GB per-machine memory usage of SRB is very visible to other services.

We explain why similarity-guided local search provides such an improvement as follows. There are VMs in which data segments are moved to another location on disk, for example when a file is rewritten rather than modified in place, and a dirty-bit or offset-only based detection would not be able to detect such a movement. We have found that in approximately 1/3 of the VMs in our dataset this movement happens frequently. Compared a previous approach [31] which maintains additional offset dirty bits within each segment, local similarity-guided search adds additional deduplication efficiency about 3%, which is a saving of 300GB per machine.

Comparing the experiment results of the sampled index method [13], our scheme achieves a 85K:1 ratio between raw snapshot data and memory usage with 96% deduplication efficiency while the sampled index method achieves 20K:1 (25GB memory per 500TB raw data) with 97% deduplication efficiency. Thus our scheme is more memory efficient with a good tradeoff for deduplication efficiency. If we change $\sigma = 4\%$, the deduplication efficiency of VC increases to 96.58% while memory usage per machine increases to about 220MB. Notice also that the sampled index method is designed for single-machine deduplication.

B. More on Resource Usage and Processing Speed

We show additional experimental results on resource usage, processing time and throughput of VC.

Storage cost of replication. When the replication degree of both PDS and non-PDS data is 3, the total storage for all VM snapshots in each physical machine takes about 3.065TB on average before compression and 0.75TB after compression. Allocating one extra copy for PDS data only adds 7GB in total per machine. Thus PDS replication degree 6 only increases the total space by 0.685% while PDS replication degree 9 adds 1.37% space overhead, which is still small.

Tasks	CPU	Mem (MB)	Read (MB/s)	Write (MB/s)	Backup Time (hrs)
1	19%	118	50	16.4	1.31
2	35%	132	50	17.6	1.23
4	63%	154	50	18.3	1.18
6	77%	171.9	50	18.8	1.162

TABLE IV
RESOURCE USAGE OF CONCURRENT BACKUP TASKS AT EACH PHYSICAL MACHINE WITH $\sigma = 2\%$ AND I/O THROTTLING.

Memory usage with multi-VM processing and disk bandwidth with I/O throttling. We have further studied the memory and disk bandwidth usage when running concurrent VM snapshot backup on each machine with $\sigma = 2\%$. Table IV gives the resource usage when running 1 or multiple VM backup tasks at the same time on each physical machine. Each task handles the backup of one VM snapshot including deduplication. "CPU" column is the percentage of a single core used. "Mem" column includes $\sim$100MB memory usage for PDS index and other space cost for executing deduplication tasks such as recipe metadata and cache. The "Read" column is controlled to 50MB/s local disk bandwidth usage with I/O throttling so that other cloud services are not significantly impacted. The peak raw storage read performance is about 300MB/s and we only use 16.7% with this collocation consideration. "Write" is the I/O write usage of QFS; note that each QFS write triggers disk writes in multiple machines due to data replication. 50MB/s dirty segment read speed triggers about 16.4MB/s disk write for non duplicates with one backup task.

Table IV shows that when each machine conducts backup one VM at a time, the entire cluster backup completes in 1.31 hours. Since there are about 25 VMs per physical machine, we could execute more tasks in parallel at each machine. But adding more backup concurrency does not shorten the overall time significantly in this case because of the controlled disk read bandwidth usage. When I/O throttling is not used, we will show below that processing multiple VMs concurrently does improve the throughput of backup.

Processing time breakdown without I/O throttling. Table V shows the average time breakdown for processing a dirty VM segment in milliseconds under VC and SRB. VC uses $\sigma = 2\%$. The overall processing latency of SRB is about 23.9% slower than VC. For VC, the change of σ does not significantly affect the overall backup speed as PDS lookup takes only a small amount of time. It has a breakdown of processing time. "Read/Write" includes snapshot reading and writing from disk, and updating of the metadata. "Network" includes the cost of transferring raw and meta data from one machine to another during snapshot read and write. "Index Lookup" is the disk,

Algorithm	Time Spent in Task (ms)		
	Read/Write	Network	Index Lookup
SRB	73	17.078	20.098
VC $\sigma = 2\%$	66.328	16.626	5.784

TABLE V
AVERAGE TIME IN MILLISECONDS TO BACKUP A DIRTY 2MB VM SEGMENT UNDER SRB AND VC WITH I/O THROTTLING.

Concurrent backup tasks per machine	Throughput without I/O throttling (MB/s)		
	Backup	Snapshot Store (write)	QFS (write)
1	1369.6	148.0	35.3
2	2408.5	260.2	61.7
4	4101.8	443.3	103.1
6	5456.5	589.7	143.8

TABLE VI
THROUGHPUT OF SOFTWARE LAYERS PER MACHINE UNDER DIFFERENT CONCURRENCY AND WITHOUT I/O THROTTLING.

network and CPU time during fingerprint comparison. This includes PDS data lookup for VC and index lookup from disk in SRB. The network transfer time for VC and SRB is about the same, because the amount of raw data they transfer is comparable. SRB spends slightly more time for snapshot read/write because during each snapshot backup, SRB involves many small bins, while VC only involves few containers with a bigger size. Thus, there are more opportunities for I/O aggregation in VC to reduce seek time. SRB also has a higher cost for index access and fingerprint comparison because most chunk fingerprints are routed to remote machines for comparison while VC handles most chunk fingerprints locally.

Throughput of software layers without I/O throttling. Table VI shows the average throughput of software layers when I/O throttling is not applied to control usage. The "Backup" column is the throughput of the backup service per machine. "Snapshot store" is the write throughput of the snapshot store layer and the significant reduction from this column to "Backup" column is caused by deduplication. Only non-duplicate chunks trigger a snapshot store write. Column "QFS" is the write request traffic to the underlying file system after compression. For example, with 148 MB/s write traffic to the snapshot store, QFS write traffic is about 35.3 MB/s after compression. However, the underlying disk storage traffic will be three times greater with replication. The result shows that the backup service can deliver up to 5.46 GB/s per machine without I/O restriction under 6 concurrent backup tasks. For our dataset, each version of total snapshots has abut 1TB per machine for 25 VMs and thus each machine would complete the backup in about 3.05 minutes. With a higher disk storage bandwidth available, the above backup throughput would be higher. In comparison, the sampled index method [13] achieves about 5.5GB per second with 16 concurrent backup tasks and 6GB per second with more tasks. Thus the per-machine throughput of our scheme is reasonable.

C. Snapshot Availability

We demonstrate the snapshot availability of VC and SRB-based VO when there are failed machines. Appendix III-B provides a detailed analysis of snapshot availability and the estimated result depends on how frequent a duplicate chunk is shared among snapshots of VMs. The trace driven execution allows us to estimate the number of file blocks shared by each VM in the VO and VC approaches and then calculate the average availability of VM snapshots.

Table VII shows the estimated availability of VM snapshots when there are up to 20 machines failed in a cluster. The case for a 1000-node cluster is also listed, assuming file block sharing patterns among VMs remain the same from $p = 100$ setting to $p = 1000$. Each machine hosts about 25 VMs in both cases and with different p values, the availability varies when the number of failures in the cluster changes.

Table VII shows that VC has a significantly higher availability than VO. With $p = 100$ and 3 failures, VO with 69.5% availability could lose data in 763 VMs while VC with 99.7% only loses data for 8 VMs out of 2500 VMs. The key reason is that for most data in VC, only a single VM can be affected by the loss of a single file block while in VO, the loss of a single block tends to affect many VMs. The availability of PDS data is very high as seen in Table II with more failures. On the other hand, the increasing of failures gradually affects the availability of non-PDS data. Overall speaking, VC can tolerate more failures compared to VO. When the number of physical machines failed reaches 20, the availability of VO reaches 0% and VC also drops to 1.13%. This is because even though availability for PDS data remains good, VC still loses many of the non-PDS blocks given the replication degree for non-PDS is 3. When $p = 1000$, the percentage of failures is then 2% and VC delivers a meaningful availability with 99.62%, outperforming VO significantly.

Figure 8 shows the impact of increasing PDS data replication degree. While the impact on storage cost is small (because we have separated out only the most popular blocks), a replication degree of 6 has a significant improvement over 4. The availability does not increase much when increasing r_c from 6 to 9 and the benefit is minimal after $r_c > 9$. That is because when the number of failed machines increases beyond 6, the non-PDS data availability starts to deteriorate significantly given its replication degree r is set to 3. Thus when $r = 3$, the reasonable choice for r_c would be a number between 6 and 9.

Failures (d)	VM Snapshot Availability(%)			
	$p = 100$		$p = 1000$	
	VO	VC	VO	VC
3	69.548605	99.660194	99.964668	99.999669
5	2.647527	96.653343	99.647243	99.996688
10	0	66.246404	95.848082	99.96026
20	0	1.132713	66.840855	99.623054

TABLE VII
AVAILABILITY OF VM SNAPSHOTS FOR VO ($r = 3$) AND VC ($r_c = 6$, $r = 3$).

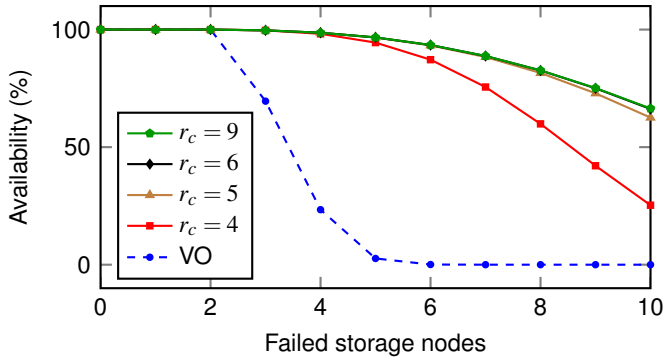


Fig. 8. Availability of VM snapshots in VC with different PDS replication degrees on a $p = 100$-node cluster.

VI. Discussions and Conclusions

The main contribution of this paper is the development and analysis of a low-profile and VM-centric deduplication scheme that uses a small amount of system resource in both snapshot deduplication and deletion while delivering competitive deduplication efficiency. The VM-centric design also allows an optimization for better fault isolation and tolerance. Our evaluation has the following results.

- *Competitive deduplication efficiency with a tradeoff*. The proposed VC scheme can accomplish over 96% of what complete global deduplication can do. In comparison, SRB [8], [3] can accomplish 97.86% while the sampled index [13] can deliver 97% for a different test dataset. Local similarity search removes 12.05% of the original data after using a segment-based dirty bit method and among them, 3% is contributed by the increased similarity search. Notice that our earlier study [31] reaches 92% of what global deduplication can do with a relatively simpler strategy and thus the new design in this paper makes a VM-centric scheme much more competitive to a traditional VM-oblivious approach.
- *Lower resource usage in deduplication*. The VC scheme achieves a 85K:1 ratio between raw snapshot data and memory usage and is much more memory-efficient than SRB with 4K:1 ratio and sampled index with 20K:1. VC with 100 machines takes 1.31 hours to complete the backup of 2,500 VMs using 19% of one core and 16.7% of IO bandwidth per machine. Processing time of VC is 23.9% faster than SRB in our tested cases and the per-machine throughput of VC is reasonable based on the result from [13]. Noted that both VC and SRB are designed for a distributed cluster architecture while the sampled index method is for a single-machine architecture.
- *Higher availability*. The snapshot availability of VC is 99.66% with 3 failed machines in a 100-machine cluster while it is 69.54% for SRB or a VM-oblivious approach. The analysis shows that the replication degree for the popular data set between 6 and 9 is good enough when the replication degree for other data blocks is 3, and adds only a small cost to storage.

The offline PDS recomputation does require some modest I/O and memory resource and since the recomputing frequency is relatively low (e.g. on a monthly basis), we expect such resource consumption is acceptable in practice. The erasure code has been shown to be effective to improve reliability on duplicated data storage [16] and such a technique can be incorporated. Other interesting future work includes studying the physical layout of backup images [25] and further assessing the benefit of PDS separation.

Acknowledgments

We would like to thank Yujuan Tan and the anonymous referees for their valuable comments, Hao Jiang, Xiaogang Li, Yue Zeng, Weicai Chen, and Shikun Tian from Alibaba for their kind support and feedback. Wei Zhang has received internship support from Alibaba for VM backup system development. This work was supported in part by NSF IIS-1118106. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or Alibaba.

Appendix A
Impact of Top k Value on Deduplication Efficiency

Choosing the top k value for the most popular chunks affects the deduplication efficiency. We analyze this impact based on the characteristics of the VM snapshot traces studied from application datasets. Additional parameters used in this analysis are defined in Table VIII. The popularity of data chunks after local deduplication follows a Zipf-like distribution[5] and its exponent α is ranged between 0.65 and 0.7.

k	The number of top most popular chunks selected for deduplication
c	The total amount of data chunks in a cluster of VMs
c_u	The total amount of unique fingerprints after perfect deduplication
f_i	The frequency for the ith most popular fingerprint
δ	The percentage of duplicates detected in local deduplication
σ	The percentage of unique data belonging to PDS

TABLE VIII
Parameters for modeling deduplication.

By Zipf-like distribution, $f_i = f_1/i^\alpha$. The total number of chunks in our backup storage which has local duplicates excluded is $c(1-\delta)$, this can be represented as the sum of each unique fingerprint times its frequency:

$$f_1 \sum_{i=1}^{c_u} \frac{1}{i^\alpha} = c(1-\delta).$$

Given $\alpha < 1$, f_1 can be approximated with integration:

$$f_1 = \frac{c(1-\alpha)(1-\delta)}{c_u^{1-\alpha}}.$$

Thus putting the k most popular fingerprints into PDS index can remove the following number of chunks during global deduplication:

$$f_1 \sum_{i=1}^{k} \frac{1}{i^{\alpha}} \approx f_1 \int_1^k \frac{1}{x^{\alpha}} dx \approx f_1 \frac{k^{1-\alpha}}{1-\alpha} = c(1-\delta)\sigma^{1-\alpha}.$$

Deduplication efficiency of the VC approach using top k popular chunks is the percentage of duplicates that can be detected:

$$E_c = \frac{c\delta + c(1-\delta)\sigma^{1-\alpha}}{c - c_u}. \quad (1)$$

We store the PDS index using a distributed shared memory hash table such as Memcached and allocate a fixed percentage of memory space per physical machine for top k popular items. As the number of physical machines (p) increases, the entire cloud cluster can host more VMs; however, ratio σ which is k/c_u remains a constant because each physical machine on average still hosts a fixed constant number of VMs. Then the overall deduplication efficiency of VC defined in Formula 1 remains constant. Thus the deduplication efficiency is stable as p increases as long as σ is a constant.

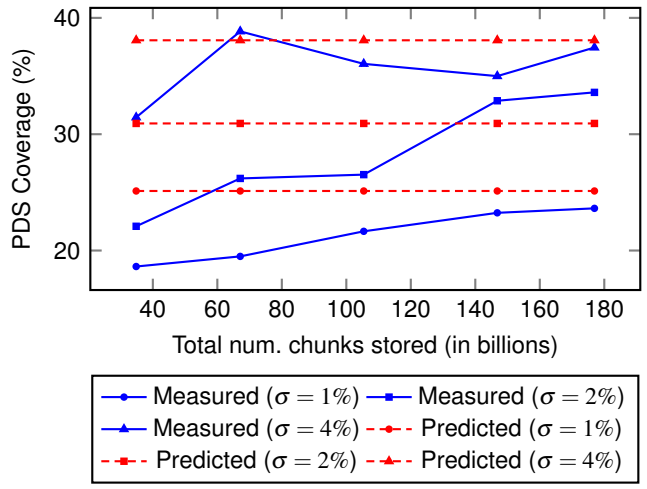


Fig. 9. Predicted vs. actual PDS coverage as data size increases.

Ratio $\sigma^{1-\alpha}$ represents the percentage of the remaining chunks detected as duplicates in global deduplication due to PDS. We call this PDS coverage. Figure 9 shows predicted PDS coverage using $\sigma^{1-\alpha}$ when α is fixed at 0.65 and measured PDS coverage in our test dataset. $\sigma = 2\%$ represents memory usage of approximately 100MB memory per machine for the PDS. While the predicted value remains flat, measured PDS coverage increases as more VMs are involved. This is because the actual α value increases with the data size.

REFERENCES

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

[2] L. Aronovich, R. Asher, E. Bachmat, H. Bitner, M. Hirsch, and S. T. Klein. The design of a similarity based deduplication system. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, SYSTOR '09, pages 6:1–6:14, New York, NY, USA, 2009. ACM.

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

[4] D. Bhagwat, K. Pollack, D. D. E. Long, T. Schwarz, E. L. Miller, and J.-F. Paris. Providing High Reliability in a Minimum Redundancy Archival Storage System. In *MASCOTS'06*, pages 413–421. IEEE.

[5] L. Breslau, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: evidence and implications. In *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies.*, pages 126–134 vol.1.

[6] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the Web. *Comput. Netw. ISDN Syst.*, 29(8-13):1157–1166, 1997.

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

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

[9] M. Y. Eltabakh, Y. Tian, F. Özcan, R. Gemulla, A. Krettek, and J. McPherson. Cohadoop: Flexible data placement and its exploitation in hadoop. *Proc. VLDB Endow.*, 4(9):575–585, June 2011.

[10] 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 *Proceedings of the 13th USENIX Conference on File and Storage Technologies, FAST 2015, Santa Clara, CA, USA, February 16-19, 2015*, pages 331–344, 2015.

[11] 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.

[12] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *SOSP'03*, pages 29–43. ACM.

[13] F. Guo and P. Efstathopoulos. Building a high-performance deduplication system. In *ATC'11*. USENIX.

[14] K. Jin and E. L. Miller. The effectiveness of deduplication on virtual machine disk images. In *SYSTOR'09*. ACM.

[15] E. Kave and T. H. Khuern. A Framework for Analyzing and Improving Content-Based Chunking Algorithms. Technical Report HPL-2005-30R1, HP Laboratory, 2005.

[16] X. Li, M. Lillibridge, and M. Uysal. Reliability analysis of deduplicated and erasure-coded storage. *SIGMETRICS Perform. Eval. Rev.*, 38(3):4–9, Jan. 2011.

[17] 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.

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

[19] Nutanix. Nutanix Complete Cluster, A Technical Whitepaper , 2013.

[20] M. Ovsiannikov, S. Rus, D. Reeves, P. Sutter, S. Rao, and J. Kelly. The quantcast file system. *Proc. VLDB Endow.*, 6(11):1092–1101, Aug. 2013.

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

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

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

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

[25] S. Smaldone, G. Wallace, and W. Hsu. Efficiently storing virtual machine backups. In *5th USENIX HotStorage (Hot Topics in Storage and File Systems), San Jose, CA, USA, June 27-28, 2013*, 2013.

[26] 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.

[27] 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.

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

[29] A. Warfield, S. Hand, K. Fraser, and T. Deegan. Facilitating the development of soft devices. page 22. USENIX.

[30] J. Wendt. A Candid Examination of Data Deduplication. White Paper, Symantec Corporation , 2009.

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

[32] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data