BigArchive: An In-Cloud Backup System With Deduplication for VM Snapshots

Wei Zhang
CS Dept.
UC Santa Barbara
wei@cs.ucsb.edu

Hong Tang
Aliyun Inc.
Alibaba Group
hongtang@alibaba-inc.com

Tao Yang
CS Dept.
UC Santa Barbara
tyang@cs.ucsb.edu

ABSTRACT

This paper proposes a VM snapshot storage architecture which adopts multiple-level selective deduplication to bring the benefits of fine-grained data reduction into cloud backup storage systems. In this work, we describe our working snapshot system implementation, and provide early performance measurements for both deduplication impact and snapshot operations.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

Keywords

ACM proceedings, LaTeX, text tagging

1. INTRODUCTION

We have designed and implemented BigArchive, a snapshot storage system to meet the rapidly growing demands of data backup from Aliyun's VM cloud users. BigArchive shares many of the same goals as previous deduplication backup systems such as storage efficiency, throughput, scalability. However, its design has been driven by key observations of our VM data duplication pattern, and resource requirements of cloud environment, that reflect a different situation from some earlier enterprise backup systems. We have reexamined traditional choices and explored radically different points in the design space.

First, data duplication between VM users are servere because of common software data. *this is the reason of needing global dedupe, why is this different from traditional backups environment?*

Second, resources allowed for our deduplication system are very limited. *why can't use 3rd party backup server, use special hardware, or a lot of memory?*

Third, the amount of small data objects after deduplication has far beyond any current storage system's capacity. *why can't gfs, bigtable store this directly?*

Fourth, the degree of data sharing must be under control. *why we must avoid all-to-all data dependency? fault isolation.*

2. DESIGN OVERVIEW

In this section we briefly describe all the static things in our system, such like architecture, each components data structure, data locations, relationship of data references, etc.

2.1 Design Considerations

We discuss the characteristics and main requirements for VM snapshot backup in a cloud environment. which are different from a traditional data backup.

1. *Cost consciousness.* There are tens of thousands of VMs running on a large-scale cluster. The amount of data is so huge such that backup cost must be controlled carefully. On the other hand, the computing resources allocated for snapshot service is very limited because VM performance has higher priority. At Aliyun, it is required that while CPU and disk usage should be small or modest during backup time, the memory footprint of snapshot service should not exceed 500MB at each node.
2. *Fast backup speed.* Often a cloud has a few hours of light workload each day (e.g. midnight), which creates an small window for automatic backup. Thus it is desirable that backup for all nodes can be conducted in parallel and any centralized or cross-machine communication for deduplication should not become a bottleneck.
3. *Fault tolerance.* The addition of data deduplication should not decrease the degree of fault tolerance. It's not desirable that small scale of data failure affects the backup of many VMs.

There are multiple choices in designing a backup architecture for VM snapshots. We discuss the following design options with a consideration on their strengths and weakness.

1. *An external and dedicated backup storage system.* In this architecture setting, a separate backup storage system using the standard backup and deduplication techniques can be deployed [3, 1, 2]. This system is attached to the cloud network and every machine can periodically transfer snapshot data to the attached backup system. A key weakness of this approach is communication bottleneck between a large number of machines in a cloud to this centralized service. Another weakness is that the cost of allocating separate resource for dedicated backup can be expensive. Since most of backup data is not used eventually, CPU and memory resource in such a backup cluster may not be fully utilized.

2. *A decentralized and co-hosted backup system with full deduplication.* In this option, the backup system runs on an existing set of cluster machines. The disadvantage is that even such a backup service may only use a fraction of the existing disk storage, fingerprint-based search does require a significant amount of memory for fingerprint lookup of searching duplicates. This competes memory resource with the existing VMs.

   Even approximation [1, 2] can be used to reduce memory requirement, one key weakness the hasn't been addressed by previous solutions is that global content sharing affects fault isolation. Because a content chunk is compared with a content signature collected from other users, this artificially creates data dependency among different VM users. In large scale cloud, node failures happen at daily basis, the loss of a shared block can affect many users whose snapshots share this data block. Without any control of such data sharing, we can only increase replication for global dataset to enhance the availability, but this incurs significantly more cost.

With these considerations in mind, we propose a decentralized backup architecture with multi-level and selective deduplication. This service is hosted in the existing set of machines and resource usage is controlled with a minimal impact to the existing applications. The deduplication process is first conducted among snapshots within each VM and then is conducted across VMs. Given the concern that searching duplicates across VMs is a global feature which can affect parallel performance and complicate failure management, we only eliminate the duplication of a small but popular data set while still maintaining a cost-effective deduplication ratio. For this purpose, we exploit the data characteristics of snapshots and collect most popular data. Data sharing across VMs is limited within this small data set such that adding replicas for it could enhance fault tolerance.

2.2 Architecture Overview

2.3 Block Storage Device

2.4 CDS

Although locality based data reduction can remove most of the inner-VM duplications, it's not able to solve the data duplication cross VMs. Different VMs still share large amount of common data such that their snapshot backups would have a lot of data in common. Our observations on Aliyun's real VM user data reveals several major sources of cross-VM data duplication:

1. *System-related data*: These data are generated by public third-parties, they are copied/downloaded/installed into VM disks either automatically or manually. Once installed, operations on such data are mostly read only until software updates arrive. For example, data of operating systems, some widely-used software such as Apache and MySQL, and their documations all fall into this category.

2. *User-generated data*: These data are generated by individual user's behavior, either directly or indirectly. They are much less duplicated than the system-related data, but the zipf-like distribution indicates that a small amount of data in this category could represent most of data duplications.

3. *Zero-filled data*: Like previous studies [refs here], we've observed that zero-filled data exist pervasively at system wide. They are almost like the spaces and commas in text articles. Under content-based chunking, they were distilled as zero-filled blocks with the maximum allowed length.

Without eliminating the data duplication between VM snapshot backups, storage space is severely wasted when the number of VMs increases. To address this problem, we developed a technique called *Common Data Set (CDS)* to eliminate data duplication for all three situations above. CDS is a public data library that shared by all VM snapshot backups in the cluster, which consists of the most popular data blocks in VM snapshot backups. It is generated, managed and accessed in an uniform manner by all VMs such that [write some fancy system characteristic stuff here].

2.5 CDS Size V.S. Coverage

As a shared data library, we expect CDS to collect almost all the system related data, the most popular user-generated data and the zero-filled data block. With CDS as a public data reference, each VM's snapshot backup has no need to store its own copies for data that can be found in CDS, instead they can just store a reference.

The more data we put into CDS, the closer we come to complete deduplication. But in reality we are facing a list of restrictions such like [blabla]. We borrowed the idea of web caching[refs here] to analysis the size of CDS vesus its effectiveness on data reduction.

We let M be the number of nodes in a cluster, N be the number of VMs hosted per node, and let D denotes the average size of one VM's snapshot backups (after level 1 and 2 reduction, which is near 40 GB in our production environment). And let C_0, C_s and C_u denote the size of zero-filled block, system-related data and user-generated data in CDS, then the total size of CDS is represented by $C = C_0 + C_s + C_u$. We let S_0, S_s and S_u denote the corresponding data coverage ratio (with respect to D) by C_0, C_s and C_u, so the total space saving ratio is $S = S_0 + S_s + S_u$.

Zero-filled block at maximum length is the first item we need to put into CDS. This is the one and only special data block, and because CDS only stores unique data blocks, C_0 is almost equal to 0. In practice we found zero-filled blocks account for near 20% of total data, so S_0 is 20%.

The rarely modified system-related data are the second to be added to CDS. We have thousands of VMs in each cluster, but all these VMs fall into only a few OS types, using a limited selection of software, therefore data duplication in this category is highly noticable in a global block counting. In particular, most of such data already exist in the VM base images. We analyzed VMs running various services from 7 mainstream OS types in our cluster, and found close to 50GB of such data in total. Because system-related data don't change frequently, it's safe to expect that for 20 OS variations and software updates in 2 years, C_s will not grow to exceed 200 GB. For each VM, from 2 to 20 GB of data can be reduced in this way, depends on the OS and service type, so on average we estimate S_s would be no less than 20%.

The rest of data are user-generated, the total size of such data can be written as $D_u = D - S_0 - S_s$, which is 60% of D. It follows a zipf-like distribution with α between 0.65 to 0.7. Let T_u be the total size of unique data blocks in D_u, in practice we notice that complete deduplication for such data always result in a 2:1 reduction ratio, , so $T_u = D_u/2$. Since we collect the most popular user-generated data into CDS, the coverage of CDS on user-generated data is $(C_u/T_u)^{1-\alpha}$.

The following table lists CDS coverage on user-generated data under different α and C_u/T_u.

[a table of coverage with alpha from 0.65-0.70, ratio from 0.002 0.005 0.01 0.02 0.05]

It's obviously that at least 30% of user-generated data can be reduced in this way, using about 0.02 of T_u, or 0.01 of D_u. The size of user-generated data in CDS would be $0.006D$, with coverage $S_u = 0.18D$.

Thus the estimation of CDS coverage is $S = S_0 + S_s + S_u = 0.58$, with the size of CDS to be no more than $(200 + 0.006 * D * N)$ GB. In a VM cluster of 100 machines, with 25 VMs on each physical machine, the size of CDS is 800 GB in total, or 8 GB per machine. The total size of CDS index would be 8 GB, which will cost 80 MB memory on each machine.

2.6 Append Store

2.6.1 Assumptions

In designing a data store for large number (billions) of small objects, we have been guided by assumptions that offer both challenges and opportunities. We alluded to some key observations earlier and now lay out our assumptions in more details.

* This system is built upon existing GFS-like file systems.
* The data store manages large number of small variable-sized objects.
* The workloads primarily consists of two kinds of reads: large straming reads and medium portion of random reads. *Excellent sequential access (scan) and good random access performance, efficient way to locate a data item*
* The workloads also have many large, sequential writes that append data to files.
* The system is implemented as a client so that no need to worry about concurrent writes.

3. SNAPSHOT OPERATIONS

4. APPEND STORE

5. CDS ANALYSIS

In this section, we need to illustrate the key observation on data duplicate pattern, this fact lead to our 3-level dedupe scheme, then combined with system requirements, they are the footstones of our system architecture. Our overall system design is based on the observation, we have reexamined traditional choices and explored radically different points in the design space.

6. FAILOVER

7. EXPERIMENTS

8. CONCLUSION

9. ADDITIONAL AUTHORS

10. REFERENCES

[1] D. Bhagwat, K. Eshghi, D. D. E. Long, and M. Lillibridge. Extreme Binning: Scalable, parallel deduplication for chunk-based file backup. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems, 2009. MASCOTS '09. IEEE International Symposium on*, pages 1–9, 2009.

[2] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezis, and P. Camble. Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality. In *FAST*, pages 111–123, 2009.

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