High-performance Disk Imaging With Deduplicated Storage

Raghuveer Pullakandam[1] *Xing Lin*[1] *Mike Hibler* *Eric Eide*
Robert Ricci
School of Computing, University of Utah

Abstract

Infrastructures such as clouds and network testbeds need to provide a large collection of "disk images". These disk images require significant resources to store. On the other side, these disk images need to be distributed to physical hosts and installed quickly. To satisfy both of these requirements, we design a new storage service to improve the storage efficiency for storing disk images, with a specific goal to distribute and install them quickly. "Data deduplication" is the technique that we utilize to reduce storage consumption. We also discuss several design tradeoffs we make for a fast disk image distribution. Evaluations of our new system demonstrate that deduplication saves more than 60% of disk space and by keeping the overall disk imaging pipeline balanced, our new system introduces a negligible performance overhead.

1 Introduction

Various virtual machine images or operating system images are provided in infrastructures such as clouds and network testbeds. The process to distribute and install disk images on physical hosts is referred to as disk imaging in the literature. The time this process takes affects user experience directly and as a result, an efficient disk imaging system is a critical component in such infrastructures.

Images in clouds or network testbeds consume large amounts of storage. For example, Emulab [5] has 602 GB of compressed images, and EC2 [1] lists more than 1,000 public Amazon Machine Images (AMIs). However, many images are slightly different versions of an operating system (eg. with different updates applied), and many are user images that are based on standard images, and thus share a lot of data with them. Previous work [3] has evaluated the effectiveness of using deduplication for *storing* virtual machine images. We go beyond storage to build a system for *distribution and installation* of disk images.

We have designed a disk imaging system that integrates the Venti [4] archival storage system into the Frisbee [2] disk imaging system. Frisbee's disk imaging mechanism was designed as a pipeline of four stages: image read from disks at the server, image distribution across a network and image decompression and disk writes at client machines. In our new design, images are stored in Venti and they are dynamically constructed on-demand. By balancing the disk imaging pipeline carefully, we show that the additional image construction time does not result in a significant drop in the overall performance of Frisbee.

2 Design and Implementation

2.1 Original Design of Frisbee

A key observation of the original Frisbee work was that the client disk is the bottleneck. The server storage system is assumed to be faster than that of the clients, and network bandwidth is minimized by the use of multicast and by the construction of Frisbee disk images.

Frisbee disk images contain only the allocated data from a disk, where "allocated" is determined by understanding the format of the filesystem(s) contained on the disk. The allocated data is read in and compressed with zlib sequentially to construct self-describing 1 MB "chunks". A disk image is composed of a sequence of chunks. By saving only the allocated data, not only are the images smaller, but we also reduce the amount of data that must be written to the slow client disk since unallocated areas are skipped. Thus, it is not unusual for a Frisbee disk image to be 10-20x smaller than the apparent size of the disk.

Since each chunk is self-describing, they may be requested and processed by clients individually and in any order, meaning that clients loading the same image do

[1] Students; Xing Lin will present

not have to operate "in lock step." In particular, clients may join and leave an image loading session at any time and the clients may be widely disparate in their processing power and disk throughput speed.

The implication for the Frisbee image server is that it must be able to handle "chunk" requests for a disk image in any order and handle redundant requests for any chunk over time.

2.2 Deduplicated Storage in Frisbee

To study the feasibility of using deduplication storage systems as the backend of a high-performance disk imaging system such as Frisbee, we chose Venti as a case study. Venti stores only a single copy of each unique block stored, and blocks are addressed by the hash of their contents. Deduplication must be done in small block sizes but Frisbee requires 1MB chunks. So, to use Venti as the backend for Frisbee, we must construct these 1MB chunks dynamically.

Deduplication based on Frisbee disk images would lead to poor deduplication ratios: Frisbee packs allocated data contiguously and then do compressions. Because of that, modifications to data near the beginning of the image would cascade to the rest of the image. To maximize the opportunities for deduplication, deduplication is done based on uncompressed disk data and boundaries of deduplication units are aligned based on disk block offsets to ensure that data occurring at the same location in multiple images will form the same deduplication units.

To minimize the performance overhead in chunk construction, we apply several optimizations. We decide to compress each deduplication unit before storing them into Venti and Venti is disabled to do compression. The main benefit is that Frisbee does not need to do expensive compressions during chunk construction. Instead, pre-compressed data retrived from Venti can be simply concatenated to build a chunk. Essentially, we moved expensive compressions from the image distribution path to the image storage path.

In order to construct any particular chunk dynamically, we have to know the set of deduplication units which are compressed to be the payload of a chunk. Determining which deduplication units are contained in a chunk fundamentally requires starting at the beginning of the image: because the compression of each deduplication unit results in highly variable compression ratios, predicting chunk boundaries is problematic. So, our design trades off higher cost at image storage time for lower cost at distribution time: when we store an image into Venti, we pre-compute all chunk headers (effectively, by creating the compressed version of the image by compressing each aligned deduplication unit). The SHA-1 hashes of stored blocks, which are used to retrieve them from Venti, are also stored as metadata. Whenever we receive a chunk request from the Frisbee server, we look up the particular chunk header to get the set of deduplication units. We retrieve these compressed deduplication units from Venti and concatenate them to form the payload of that chunk. Appending the payload after the chunk header, the complete chunk is returned to the Frisbee server. This design allows us to independently construct any chunk required by Frisbee.

3 Evaluation

Our evaluation focuses on the storage saving and the impact on the overall performance of disk imaging. 32KB is used as the size for deduplication units.

Storage Savings: Using the set of 430 Linux disk images collected by Emulab, Venti deduplicates 69.312% of it. Adding the metadata into consideration, Venti achieves 67.82% end-to-end disk saving.

Impact on The Overall Disk Imaging Performance: We compared the time it takes to distribute the standard Emulab Ubuntu 10 image to 1 client, 8 clients and 16 clients, using the original Frisbee and our new Venti-based Frisbee. For 1 client, the original Frisbee takes 22.37 seconds while our new Frisbee takes 23.00 seconds. A 2.7% increase in the disk imaging pipeline is hardly noticable. Results for 8 clients and 16 clients are almost the same.

From the evaluation of our system, we show that using Venti as the backend to store images gives us a huge storage saving and the performance overhead is negligible. We conclude that using a deduplication storage system as the storage back-end for a high-performance disk imaging system is practical.

References

[1] AMAZON WEB SERVICES LLC. Amazon machine images (AMIs). http://aws.amazon.com/amis, Sept. 2011.

[2] HIBLER, M., STOLLER, L., LEPREAU, J., RICCI, R., AND BARB, C. Fast, scalable disk imaging with frisbee. In *Proc. of the 2003 USENIX Annual Technical Conf.* (San Antonio, TX, June 2003), USENIX Association, pp. 283–296.

[3] JIN, K., AND MILLER, E. L. The effectiveness of deduplication on virtual machine disk images. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference* (New York, NY, USA, 2009), SYSTOR '09, ACM, pp. 7:1–7:12.

[4] QUINLAN, S., AND DORWARD, S. Venti: A new approach to archival data storage. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2002), FAST '02, USENIX Association.

[5] WHITE, B., LEPREAU, J., STOLLER, L., RICCI, R., GURUPRASAD, S., NEWBOLD, M., HIBLER, M., BARB, C., AND JOGLEKAR, A. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation* (Boston, MA, Dec. 2002), USENIX Association, pp. 255–270.