CASE: Exploiting Content Redundancy for Improving Space Efficiency and Benchmarking Accuracy in Storage Emulation

Lei Tian (presenter)
University of Nebraska-Lincoln

Hong Jiang
University of Nebraska-Lincoln

1 Introduction

For decades, benchmarking file and storage systems remains a difficult task for evaluators and practitioners alike, where the obtained evaluation results are often considered inaccurate, incomplete and sometimes misleading [1, 2]. To offer realistic, interactive, comprehensive evaluations, storage emulation, a technique that allows simulated storage components to be plugged into real systems to run real applications, has become a practically feasible and attractive benchmarking solution.

To date, we observe that in practice many benchmarks and applications are very sensitive to and rely on the contents of data sets as input. They usually have very different demands for computational power and storage space depending on the data contents. Taking a commonly-used compression application *tar* as a simple storage benchmarking example, we compress three bitmap image files ("animation.bmp", "wolf.bmp", and "white.bmp" respectively, 2400×1920 resolution, file size of 13,824,054 bytes), and use a block layer tracing utility (*blktrace* to collect the detailed information of I/O requests during saving compressed files. The results shown in Figure 1 indicate that the compression times and resulting I/O patterns may significantly differentiate from each other as a function of different image contents. More importantly, without storing the exact data contents, it is highly likely for tar to report errors and then fail to complete when it is used to evaluate the compressing performance in storage benchmarking. Further, modern-day file systems have become increasingly content-sensitive since they employ content-based technologies such as checksum, crypto, and compression to satisfy the demanding requirements for high reliability, security and storage efficiency.

Therefore, we argue, to retain the exact data contents of datasets used in storage systems evaluations is a necessity for content-sensitive applications, benchmarks, and file systems. Otherwise, it will be very prone to either overestimating or underestimating the real performance of storage systems. This in turn can mislead designers and administrators to draw incorrect conclusions when such content-sensitive applications and benchmarks are used but without the input of the exact data contents. Therefore, how to provide a space-efficient storage system evaluation platform while storing the exact data contents becomes a challenging but imperative problem that must be addressed.

2 Design and Implementation

To this end, we propose *CASE*, a flexible content-aware and space-efficient storage emulator for benchmarking. The basic idea behind CASE is simple: it deploys the data deduplication technique to eliminate duplicates to achieve the goal of space saving for content-sensitive applications and benchmarks. CASE is inspired by two key observations: (1) the ubiquitous data redundancy in real-world workloads. A recent study of the workloads obtained from a virtual machine running two web servers ("web"), an email server ("mail"), and a file server ("homes"), shows that the unique writes account for 42.35%, 7.83%, and 66.37% of the total writes under the web, mail, and homes workloads respectively [3]; and (2) the very high level of data redundancy in most, if not all, existing benchmarking tools. For example, Postmark, Filebench, and IOmeter) opt to use a common data buffer to temporally store read/written data for each request. They usually initialize the data buffer with all zeros in the initialization phase, and then issue I/O requests with this buffer repetitively.

CASE is designed to be a timing-accurate block-level storage emulator beneath the standard SCSI storage protocol, while performing inline data deduplication for the space-reduction purpose that is transparent to the upper file systems and applications to be benchmarked. In particular, a pure block-level storage emulation design makes CASE independent of any upper file system and database. We can install and benchmark any type of file systems or databases on CASE without any constraint on and modification to them.

Figure 2 shows an architectural view of CASE. Beneath a SCSI target subsystem layer, CASE interprets and responds to the SCSI commands to emulate a SCSI disk. CASE consists of three main functional modules: *Request Handler*, *Timing Service*, and *Storage Service*. Request Handler is responsible for receiving and enqueuing I/O requests from the upper layer and, after processing, forwarding them to the other two modules. Request Handler is also responsible for dequeuing the I/O requests and returning the status (and the corresponding

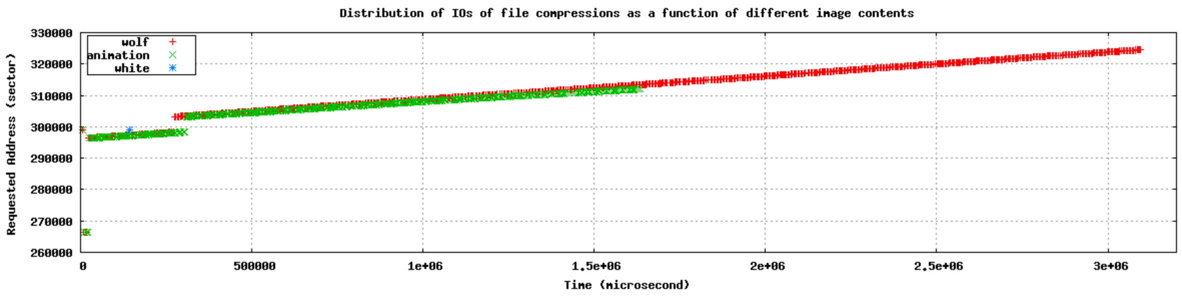

Figure 1: Distribution of IOs of file compressions as a function of different image contents.

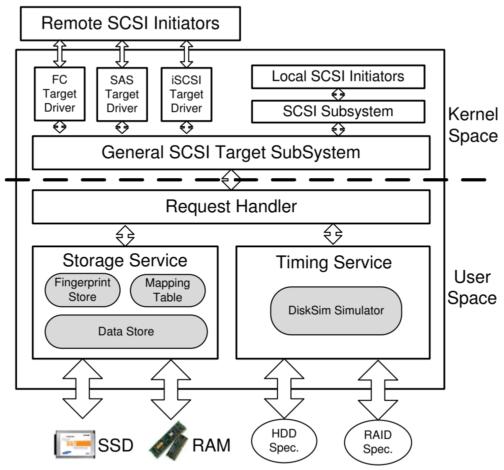

Figure 2: The CASE architecture.

data for reads) to the upper layer, after they are properly processed by the other two modules.

The function of Timing Service is to compute the response time for every incoming I/O request by simulating the I/O behaviors of the targeted storage devices with a simulation engine, and then inform CASE of the exact completion time according to the obtained time delay. Similar to Memulator [4], CASE uses the well-known storage system simulator, DiskSim [5], to simulate the target storage components and devices. We choose DiskSim as the simulation engine to provide timing control for two main reasons: 1) DiskSim has been the most commonly used storage simulator because of its calibrated device models and recognized simulation accuracy; and 2) DiskSim is highly configurable. Its hierarchical framework can support multiple layers of various storage components (e.g., buses, controllers, HDDs, SSDs, and RAIDs) and specification parameters can be tuned on demand. By integrating DiskSim into the Timing Service module, CASE is capable of performing timing-accurate storage emulation with a flexible support of various types of storage components and their combinations.

For space efficiency, CASE generates fingerprints for each write request and lookups them in a fingerprint store to eliminate duplicate writes and minimize footprints of workloads. For each read request, CASE relies on a special two-tiered address-mapping table to retrieve corresponding data from the underlying storage device, and reports the completion with the data and status.

3 Conclusion

The content-aware, albeit semantic-oblivious, feature of CASE enables it to significantly reduce storage space requirements for both content-sensitive and content-insensitive applications and benchmarks, without the substantial burden of file system semantics awareness and conveyance necessary for the semantic-aware storage emulation approaches. We prototype CASE and preliminary experimental results demonstrate its potential power. The experimental results show that it can reduce storage requirements by up to 2 orders of magnitude while reducing the write traffic by up to 85.4%.

References

[1] Avishay Traeger, Erez Zadok, Nikolai Joukov, and Charles P. Wright. A Nine Year Study of File System and Storage Benchmarking. *ACM Transactions on Storage*, 4(2):25–80, 2008.

[2] V. Tarasov, S. Bhanage, E. Zadok, and M. Seltzer. Benchmarking file system benchmarking: It *is* rocket science. In *HotOS XIII*, May 2011.

[3] Ricardo Koller and Raju Rangaswami. I/o deduplication: utilizing content similarity to improve i/o performance. In *FAST'10*, 2010.

[4] John Linwood Griffin, Jiri Schindler, Steven W. Schlosser, John C. Bucy, and Gregory R. Ganger. Timing-accurate storage emulation. In *FAST '02*, Berkeley, CA, USA, January 2002.

[5] John S. Bucy, Jiri Schindler, Steven W. Schlosser, Gregory R. Ganger, and Contributors. The disksim simulation environment version 4.0 reference manual. Technical Report CMU-PDL-08-101, Carnegie Mellon University Parallel Data Lab, May 2008.