Model Checking a Decentralized Storage Deduplication Protocol

João Paulo
University of Minho
jtpaulo@di.uminho.pt

José Pereira
University of Minho
jop@di.uminho.pt

***Abstract*—Deduplication of live storage volumes in a cloud computing environment is better done by post-processing: by delaying discovery and removal of duplicate data after I/O requests have been concluded, impact in latency can be minimized. When compared to traditional deduplication in backup systems, which can be done in-line and in a centralized fashion, distribution and concurrency lead to increased complexity. This paper outlines a deduplication algorithm for a typical cloud infrastructure with a common storage pool and summarizes how model-checking with the TLA+ toolset was used to uncover and correct some subtle concurrency issues.**

I. Introduction

Although deduplication has been in use for a long time in backup and archival systems [1] and is now an important feature of storage appliances, cloud computing in general and the commodity server infrastructure in particular bring novel opportunities, needs, and means to apply deduplication to general purpose live storage volumes. An emerging and interesting example is the redundant data found across virtual machine (VM) images, which are massively used in cloud computing infrastructures. It has been pointed out that up to 95% of space can be reclaimed for system images in a typical cloud provider.

Physical space reclaimed by deduplication allows reducing storage's infrastructure costs and having extra space for improving storage's reliability with, for example, additional RAID configurations. Recognizing duplication might also have a positive performance impact throughout the storage management stack, namely, in cache efficiency and network bandwidth consumption.

This paper presents a deduplication algorithm for a distributed environment with a shared storage that can be mapped to a typical scenario where several commodity servers are storing their data in a common pool. More specifically, the algorithm eliminates duplicated data among groups of VMs running in each server and storing their images in a virtual shared storage device. Performing deduplication in on-line infrastructures raises performance concerns that are not addressed in backup systems, namely, the I/O requests from VMs and server resources cannot be affected significantly when deduplication is being performed. Post-processing approaches for deduplication must be used to comply with these requirements; however such approaches allow for additional concurrency and thus increase the complexity of the solution. Moreover, existing decentralized deduplication proposals [2], [3] do not give any guarantee about the correctness of their algorithms, which we assume desirable in complex solutions like these ones.

The rest of this paper is as follows. Section II presents a brief description of the deduplication algorithm. Section III describes how the TLA+ toolset [4], [5] was used to find errors in the algorithm, and how these were then corrected.

II. Deduplication Algorithm

The deduplication algorithm can be divided into three main modules: I/O interception module, share module and garbage collector (GC) module. The I/O interception module intercepts VMs I/O requests (read/write block requests to virtual addresses) and maps them to the appropriate location at the storage (physical addresses). Mappings between virtual and physical addresses are kept in a metadata structure (translation table) that presents the basic mechanism to share identical physical blocks. Sharing is achieved by pointing two distinct virtual addresses to the same physical address and requires a copy-on-write mechanism that prevents updating physical blocks shared by more than one virtual address. Copy-on-write requires unused (free) blocks to write the content of the new blocks, which is achieved with additional metadata structures that track unused blocks at the storage.

The actual sharing is done by the share module in a post-processing asynchronous fashion. Periodically, virtual and physical addresses of written blocks, registered by the I/O interception module, are collected and shared. These blocks are marked as copy-on-write and their signatures are calculated with a Hash function after reading their content from the storage. Signatures are sent to a distributed hash table (DHT) used to track the unique block signatures found at the storage and the number of virtual addresses sharing those blocks. The DHT allows finding identical data in a single image of a VM and across several VMs images.

There are two possible responses from the DHT: the signature matches one of the existing entries or the entry with that signature does not exist. For the first outcome, the translation table is updated by pointing the virtual address to the physical address referred in the DHT's entry, the physical block being shared is freed and the counter for that DHT's entry is incremented. For the second response, a DHT's entry is created with the signature and address of the physical block and the counter is set to one. Locally, no actions are taken because there are no candidates to share the block.

Copy-on-write can be viewed as an operation that removes the link between a virtual address and a shared physical address that may be freed if no other virtual address is sharing it. All these operations are registered by the I/O interception module and are collected periodically and asynchronously with the GC module. This module calculates the signatures of the physical blocks with an approach identical to the one followed by the share module and sends these signatures to the DHT. The DHT decrements the entry's counter and returns it along with the physical address pointed by that entry. If the counter reaches the value zero, the physical address can be freed, otherwise, the block is still linked to other virtual addresses and cannot be freed. For this remote call the signature must always exist at the DHT because a request sent by the GC to the DHT is always preceeded by a correspondent request done by the share module.

Along this description, we referred for both share and GC modules that physical blocks can be freed. Free blocks are kept in a queue, independent for each server, that is used to provide blocks for copy-on-write operations. Since the purpose of this algorithm is to share data and free unused blocks, only a few number of necessary blocks are kept in these queues for copy-on-write, the remaining blocks are sent to a remote server that tracks all the unused blocks found at the shared storage. This server resembles the extend server of Parallax [6] solution and is used by the servers to obtain and free unused blocks.

III. Model Checking

In order to model check the proposed algorithm, it was encoded using the CAL language. Two safety properties were then specified using the TLA+ language: (i) That values read from a storage block correspond to values actually written there, thus excluding corruption; and (ii) that blocks marked as having different values in the DHT don't ever point to the same physical page. The configuration used had 2 virtual storage blocks, 3 physical disk blocks, 2 processes, and 2 block values. The problem uncovered is as follows.

Consider that the share module runs for one VM, marks a specific physical address A, which was not marked previously, as copy-on-write and starts calculating the Hash signature for that address. Concurrently, the I/O interceptor, of the same VM, receives a write request for the virtual address that is pointing to physical address A and, since the address was already marked as copy on write, it is written a copy of that address and the physical address A is ready to be collected by the GC module. In fact, if the GC module processes the address A and sends the remote request to the DHT before the share module, which is processing the same address concurrently, the counter for that signature will be decremented before the corresponding increment. This can lead to two serious problems: If the entry was not present at the DHT then no address will be decremented but eventually the share module will make its remote request and produce a dangling address that will never be freed by the GC. On the other hand, if the address is present at the DHT and after decrementing it the counter reaches zero, the address is freed which may lead to data corruption because some other virtual address was pointing to that physical address.

We can solve this problem by forcing the share and GC modules to be synchronized and running sequentially for the same VM, since this problem does not occur for the share and GC modules of different VMs. Such solution does not introduce any negative impact in performance, since no fine grained locking is required.

IV. Conclusion and Future Work

This paper presents a highly concurrent deduplication algorithm for a virtualized distributed scenario with a shared storage. Complexity is mainly introduced by the post-processing approach that is necessary to achieve interactive performance requirements. Model checking with the TLA+, even with limited scope, was found to be extremely useful in finding concurrency issues. Currently, we have a newer version of the algorithm that tolerates crash failures and has several optimizations to improve the fairness between the operations of share and I/O modules. As future work, we intend to specify and extend this work to the complete version of the algorithm.

Acknowledgments

This work is partially supported by FCT project PDTC/EIA-EIA/109044/2008.

References

[1] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage," in *FAST '02: Proceedings of the Conference on File and Storage Technologies*, 2002.

[2] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li, "Decentralized deduplication in san cluster file systems," in *Proceedings of the 2009 conference on USENIX Annual technical conference*, 2009.

[3] J. Paulo, "Efficient storage of data in cloud computing." Master's thesis, 2009.

[4] L. Lamport, *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[5] L. Lamport *et al.* (2009) Tla+ toolset. [Online]. Available: http://www.tlaplus.net/tools/tla-toolbox/

[6] D. Meyer, G. Aggarwal, B. Cully, G. Lefebvre, M. Feeley, N. Hutchinson, and A. Warfield, "Parallax: Virtual disks for virtual machines," in *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, 2008.