SimDedup: A simulator for block level deduplication in primary storage systems

Chinmay Chiplunkar
chinmayvc@gmail.com
Birla Institute of Technology and Science, Pilani- KK Birla Goa Campus
Zuarinagar, Goa, India

Biju Raveendran
biju@goa.bits-pilani.ac.in
Birla Institute of Technology and Science, Pilani- KK Birla Goa Campus
Zuarinagar, Goa, India

ABSTRACT

In this study, we propose a simulator tool SimDedup for testing the efficacy of deduplication in a primary storage environment. With the advent of cloud computing and IoT, the amount of data generated is rising fast. In this light, it is important to explore various different methods to carry out deduplication of data. With SimDedup, we lay the groundwork for a robust simulator that datacentre administrators and researchers can use to find the best deduplication solution for them. We demonstrate that SimDedup is able to accurately simulate the process of block level deduplication for different types of workloads. We conclude by proposing avenues to focus on for future research.

CCS CONCEPTS

• **Storage systems → Data deduplication**.

KEYWORDS

data deduplication, cloud computing, simulation tools

1 INTRODUCTION

There has been a massive increase in the amount of data generated globally[1]. As a large amount of computation happens in cloud computing datacenters globally, the amount of duplicate data generated and stored in the same storage cluster has been observed to be very high, exceeding even 80 percent of total stored data at times[11]. Thus, storage cost savings can become quite significant if we are able to eliminate the duplicate data and instead store just one copy of it. Data deduplication aims to achieve just this. Previous work in this area has focused on developing specific solutions for single server systems. The deduplication of data in these systems occurs at the level of the server's internal storage. A few researchers have attempted to develop deduplication tools for mutli-server systems but most often these are hardware specific. Testing out and experimenting with these solutions requires access to a large amount of servers and sophisticated storage hardware. Such hardware may not be accessible to a large number of research groups, thus making this a significant roadblock on the path to developing a robust, cloud-scale deduplication solution. Not only that, even if access to hardware of the required scale is available, due to the multitude of different approaches adopted by different solutions, it is very hard to evaluate them qualitatively. However, empirical evaluation of multiple solutions would involve making large scale changes to production server systems. In such a scenario, rolling back the changes if a particular solution is found to be undesirable or ineffective can become a tedious process which may require temporarily shutting down production servers. In light of this, there is a need to develop a simulator which researchers and cloud administrators can test their workloads on before implementing the deduplication tool in practice on their servers. The optimum deduplication system can vary based on the type of I/O workload, data redundancy requirements and the type of storage technology used. As a result we have laid the groundwork for SimDedup, a simulator inspired by the deduplication strategy first proposed by[12]. It is a fully decentralized, scalable, out-of-line deduplication solution capable of carrying out exact deduplication, which means that it can eliminate all duplicate copies of data. It is for this reason that we have adopted this solution as the basis for our simulator. SimDedup simulates all the essential features of this deduplication strategy and lays the groundwork for further enhancements. The remainder of this report is organized as follows: Section 2 provides information about the different solutions developed for distributed deduplication in primary storage systems, Section 3 elaborates on the high-level design adopted by SimDedup, Section 4 describes the implementation of the design in detail, Section 5 describes the experiments carried out to evaluate the working of SimDedup, Section 6 sheds light on the future avenues of research in this field and Section 7 describes the conclusions from our study.

2 BACKGROUND ON DEDUPLICATION

There are two main types of deduplication systems: inline and out-of-line. In inline systems, deduplication activities like block fingerprint computation, fingerprint lookup and updation of the fingerprint table and number of references to a physical block address happens on the write path. The advantage of this is that duplicate writes are prevented before they can occur. However, the process of deduplication has an impact on the write latency. Most solutions in this field of research focus on optimizations that can reduce the write latency while carrying out as much deduplication as possible. In out-of-line deduplication, all writes are allowed to take place without interruption. A background process periodically carries out deduplication of newly written blocks. While this does

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

not impact write latency in any way, it can affect the write throughput of the system if the background process takes up a significant chunk of the computational time when it executes.

Data deduplication was first explored in the context of data backup and archival systems. Since two consecutive backups are likely to have a large amount of duplicate content, this translated to significant cost savings. In backup and restore systems, data throughput is valued over latency. As a result, the deduplication solutions for such systems focus on improving both backup and restore throughput[7, 10, 17].

In the case of deduplication in primary storage systems, latency of data storage and retrieval is extremely important in order for cloud providers to satisfy their SLAs. As a result, the type of optimizations proposed are very different. Some solutions try to carry out deduplication at the level of files and hence deduplication file system based solutions like DDE[4],DEDE[3], TDDFS[2], etc. have been developed. However, these solutions are not generalized and can be used in only certain specialized settings. For example, DDE requires the use of the proprietary IBM Storage Tank File System. Similarly, DEDE has been designed for the proprietary VMWare file system VMFS. TDDFS requires a very specific storage architecture, one that contains a fast tier consisting of SSD storage and a slow tier comprising of traditional hard drives. Not only that, it was found that a lot more duplicate data exists at the block level rather than at the file level. It was found that full-file deduplication can save only about 75 percent of storage space as compared to a block level deduplication system[11].

This has led to many block level deduplication solutions being developed. Works like iDedup[13], dmdedup[14], HANDS[15] have proposed approaches that involve optimizations like preventing file fragmentation by setting a deduplication threshold, storing deduplication metadata on SSDs and implementing dynamic in-memory caches of block fingerprints for lowering write latency. While these systems act as useful case studies, their practical value is limited as, in typical cloud systems data duplication occurs on the scale of entire servers and also many servers are connected to a common storage backend. Single server solutions do not specify policies for synchronization and parallelization of the deduplication process across multiple servers and hence are not scalable.

Relatively less work has been done on deduplication in a distributed or cluster computing environment. However, it is at this level that maximum deduplication can be carried out rather than at the level of individual servers. At the scale of multiple server systems, in-line deduplication can have unacceptably high write latency due to the required inter-server communication for querying the fingerprint table. One of the proposed solutions in this area is [5], which

As a result out-of-line deduplication systems like DDE[4], DEDE[3] and DEDIS[12] have been proposed. Apart from the aforementioned solutions, hybrid solutions that aim to combine inline and out-of-line deduplication like HPDedup[16] have been also proposed. Such solutions propose to maintain a very small dynamic cache in main memory to eliminate as many duplicate copies of data on the write path and eliminating the others via out-of-line deduplication. We have chosen DEDIS as the solution of choice to implement on our simulator due to its simple, scalable, generalized and extensible design.

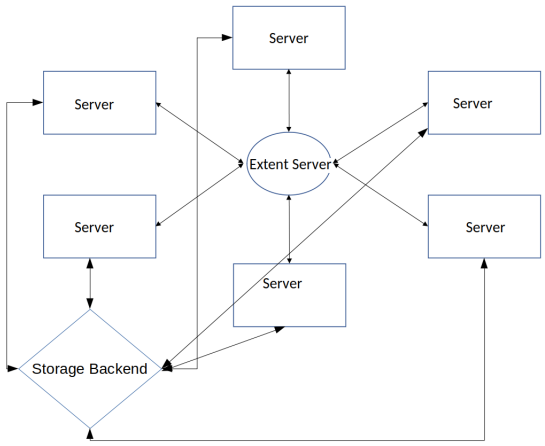

Figure 1: This figure shows an overall schematic of the main components that the SimDedup simulates. Servers are connected to both a common storage backend as well as an extent server which allocates storage individually to each server

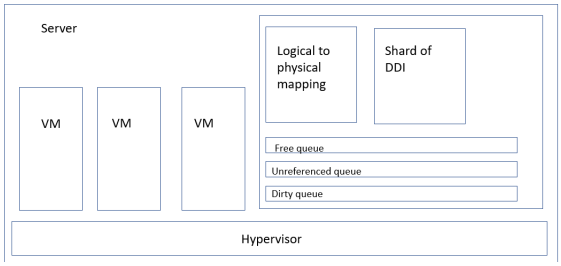

Figure 2: This figure shows a schematic representation of the internal components of a server

3 DESIGN

In this section, we will give an overview of the design of our simulator, which has been adapted from DEDIS[12].

Figure 1 shows a graphical representation of the overall design of the simulator. A cloud environment containing multiple servers connected to a common storage backend is assumed. Each server is connected to the storage backend. Also, there is an extent server whose function is to allocate blocks from the common storage backend to individual servers.

Figure 2 shows a graphical representation of the internals of a server. Each server runs multiple virtual machines(VMs). One virtual machine named Dom0 is the master VM and all accesses to memory, storage and the network occur via it. Dom0 contains three queues: free, dirty and unreferenced which store block addresses. The mapping from LBAs to physical block addresses is also stored on Dom0. Each server also stores a shard of a distributed index known as DDI(distributed duplicates index). This index is required to carry out scalable, decentralized deduplication. The overall functioning of the server can be split into three parts as follows:

- Handling a write request:

(1) Convert the logical address to physical address.
(2) If no physical address exists for that logical address, then obtain a free block from the free queue and write to it. Insert the entry for this block in the logical to physical mapping.
(3) If a physical address for the LBA already exists, then check if it is COW.
(4) If not COW, then write it in-place.
(5) If the LBA is COW, then obtain a new block from the free queue and write to it. Update the COW status and logical to physical mapping of the LBA. Insert the address of the old physical block corresponding to the LBA into the unreferenced queue.
(6) Insert the block that was written to into the dirty queue.

- Out-of-line deduplication:

(1) Get the block at the front of the dirty queue and carry out logical to physical address conversion(say block A).
(2) Read the block and calculate its signature.
(3) Determine which DDI shard may contain that signature and lock that particular shard .
(4) Query the DDI with the signature of the dirty block
(5) If there is no potential alias in the DDI, then insert a new entry into the DDI corresponding to that signature.
(6) If there is potential alias(say block B), then mark B as COW.
(7) The signature of B in the DDI may be stale. So read B and calculate its signature.
(8) If signature of A and B are the same, then update the logical to physical mapping for logical address. corresponding to block A. Also increment the number of references to B in the DDI. Put A in the free queue.
(9) If signature of A and B are not equal, then insert a new entry into the DDI corresponding to signature of A. If B was not previously marked COW, then remove COW mark of B.
(10) Unlock the DDI shard.

- Garbage collection:

(1) Get the next block address from the unreferenced queue(say A).
(2) Read block A and calculate its signature.
(3) Determine which DDI shard may contain that signature and lock that particular shard .
(4) Query the DDI shard with the signature of block A.
(5) If there is no entry in the DDI for the signature of A, then put A in the free queue.
(6) If there is a matching entry for the signature of A(say for block B), then check whether A and B are the same physical address. If they are, then decrement the number of references of B. If the number of references to B becomes 0, then remove its entry from the DDI and put it in the free queue.
(7) Unlock the DDI shard.

4 IMPLEMENTATION

SimDedup is implemented using C++ using the Standard Template Library(STL) and C wrapper libraries for UNIX system calls. The total length of the code is more than 1000 lines. The entire code is modular and contains the following key files:

- server.cpp: It contains the definition of the class Server and all of its methods. Each Server object runs in a separate thread. The important member variables of Server are as follows:

(1) the inner class VM - for each VM there is a single input file containing the LBAs it wants to write to and the MD5 hash of the write content
(2) a list of VMs running on the server
(3) id of the currently executing VM - required for making VM scheduling decisions
(4) the inner class Dom0 - represents the master VM and contains a dictionary for logical-to-physical address mapping and STL queues which represent the dirty, free and unreferenced queues.

Some of the important methods of Server are:

(1) write - This method carries out a write and uses several helper methods like checking the COW status of a block, converting from logical to physical address, getting a new block from the free queue and so on
(2) runDuplicateFinderOnNextBlock - Carries out deduplication on the block at the front of the dirty queue using the algorithm described in the previous section. It uses the helper methods of the DDI to lock the appropriate index shard, to query it and to update it if required
(3) runGarbageCollectionOnNextBlock - Carries out garbage collection on the block at the front of the unreferenced queue using the algorithm described in the previous section. It too uses the helper methods of the DDI.

- storage.cpp: It contains the definition of the class extentServer and all of its methods. The important member variables of extentServer are as follows:

(1) allocationMap - a vector that stores the id of the server to which each block of storage has been assigned
(2) contents - a vector that stores the MD5 hash of the contents of the block as a string
(3) extentServerMutex - a semaphore that is used to ensure that only one server is able to request block allocation at a time and that two servers do not get allocated the same block of storage due to concurrent allocation requests.

Some of the important methods of extentServer are:

(1) allocateExtent - used to allocate a large chunk of storage to a server which it can then utilize of smaller granularity through its free queue. The algorithm used for allocation is next fit, in which the process of searching for a chunk of sequential blocks which is larger than the request size starts from the address at which the search stopped at the previous allocation request.
(2) updateSignature - updates the MD5 hash value value stored in the contents vector

- DDI.cpp: It contains the definition of the class DDI(short for Distributed Duplicates Index) and all of its methods. The member variables of DDI are as follows:

(1) indexShards - This is a vector of dictionaries where each dictionary stores the contents of one shard of the distributed index.
(2) shardlocks - This is a vector of semaphores, where each semaphore is used to allow only one Server thread to access a shard at a time
(3) secondLevelIndex - It is used to determine which particular shard may contain the entry for a given fingerprint. Since it involves comparing the fingerprint with the lowest and highest fingerprint value allowed for each shard and MD5 hash is a 128 bit number, we need to convert it from a string to a uint128_t, which is an unsigned 128 bit integer datatype provided by the Boost C++ library.

Some of the important methods of DDI are:

(1) queryDDI - Given a fingerprint and a shard number, it is used to check whether an entry corresponding to the fingerprint is present in that shard. It utilizes helper methods to lock and unlock the semaphore corresponding to the shard.
(2) incrementDDI - It is used to increment the number of references to the physical address corresponding to a particular signature in the given shard. A similar method decrementDDI is used for the opposite purpose.
(3) insertNewEntryInDDI - Inserts a new entry corresponding to a particular fingerprint and physical address in the DDI. A similar function deleteEntryFromDDI is used while garbage collection is taking place to remove an entry from the DDI.

- driver.cpp: This is the driver module and contains the main() function. It takes inputs from a configuration file('write path here') and accordingly creates a separate thread corresponding to each server. Within each server there can be multiple VMs and each VM has its own input file of logical block addresses(LBAs) which it wants to write to. This file also contains the following global variables:

(1) ddi - A global DDI object so that all server threads can access it
(2) exServer - A global extentServer object so that all server threads can access it
(3) writeThreshold - the number of writes to carry out before carrying out deduplication and garbage collection. It is read as a parameter from the configuration file
(4) COWSignalVectorMap - this dictionary of vectors stores the COW LBAs for all servers. It is used when a block deduplication occurs and one thread must set the LBA of another thread as COW as well to avoid overwriting.

- configuration file("./inputs/input.txt"): The configuration file is structured as follows:
  - 1st line - total size of storage in number of blocks
  - 2nd line - write threshold
  - 3rd line - number of servers

  After that for each server, the number of VMs and the paths to their input files are present on a new line.
- Input VM traces: Each of these files corresponds to one VM's write requests. Each line of these files has the following structure:
  - LBA
  - R or W - read or write
  - fingerprint - 128 bit MD5 hash

5 EXPERIMENTS

In order to evaluate SimDedup, we tried running it with different VM traces with different I/O characteristics. The traces have been obtained from Florida International University's servers[6].The following traces were tried:

(1) Web VM trace: This trace is from a VM that was hosting a online course management system and an email access portal. The VM had access to 70GB of storage out of which 11.2GB was written to and 3.4GB was read from.
(2) Mail server trace: This trace is from a VM that was hosting email inboxes of several users. It had access to 500 GB of storage out of which 482.10 GB was written to and 62 GB was read from.

SimDedup was able to carry out deduplication in a similar manner to DEDIS[12], thus showing that it is an effective simulator for deduplication.

Moreover, SimDedup does not require heavy computational resources as it was tested on a Dell Inspiron personal laptop with a Intel Core i5 7th generation processor, 8 GB DDR-4 RAM and a 1 TB Seagate HDD. This makes it viable for almost all researchers to use, especially those who do not have access to an expensive setup of multiple servers connected to a storage backend.

6 FUTURE WORK

We believe that SimDedup builds a flexible, extensible framework for carrying out research on different deduplication techniques effectively and that the proposed by [12] which we have used is far from the optimal technique for deduplication in distributed storage systems. The following are the most promising avenues of research to build upon the framework of SimDedup:

- Dealing with high write latency for sequential COW writes: Allocating blocks from the free queue for large sequential COW writes cause a high penalty on the write latency. In order to avoid this, prediction mechanisms for sequential writes can be developed and the required amount of blocks can be pre-allocated for such writes. Different prediction and pre-allocation techniques need to be developed and compared for this purpose.
- Dealing with high read latency caused by file fragmentation: Due to deduplication, file blocks that would have been written sequentially become fragmented and this increases the number of disk seeks required for file reads. To avoid this, we must avoid deduplicating those sequential blocks that are likely to be read together frequently. iDedup[13] proposes setting a static threshold for the minimum sequence length of duplicate blocks whose duplicates are also stored sequentially for carrying out deduplication. If the number of sequential blocks which have sequential duplicates is lower than the threshold, then deduplication is not carried out. However, the appropriate threshold depends on the acceptable increase in read latency vs percentage of deduplication

and the resulting storage cost reduction. Also, the I/O characteristics of different workloads are likely to require different thresholds.

- Avoid deduplicating critical data such as filesystem metadata or very frequently written file blocks: The block layer does not have any file-system related information. However, previous research work has been attempted to pass hints down from the file-system layer to the block layer to make the deduplication process less naive and more refined[9]. Several different types of hints are possible to prefetch hashes, partitioning the hash index according to file type to reduce lookup time, etc. are possible. Further work is required to integrate such hints into the framework developed by SimDedup and to analyse their effectiveness with regards to different types of workloads.
- Using Bloom filters to make index lookup faster: Some studies in the past have used a technique called Bloom filters to quickly know with a certain confidence level whether a fingerprint has been seen or not[8]. This phase can be thought of as the 'prelookup' phase and is able to filter out a large number of writes which do not need to be checked for deduplication. Results can have shown that the performance of fingerprint lookup can be improved by more than 50 percent by using this method. Further work is required to ascertain whether this technique can be effectively combined with the framework of SimDedup.
- Hybrid solutions: Some systems like HPDedup[16] propose hybrid inline and out-of-line methods. The advantage of such solutions is that they are able to eliminate many writes on the write path at an acceptable memory overhead. This helps to save on storage costs while maintaining an acceptable write latency. The remaining duplicates can be eliminated by an out-of-line deduplication process that does not impact system throughput as much as one in which no inline deduplication occurs. However, implementing low latency cache coherence policies for such systems in multi-server environments is a challenging problem.

7 CONCLUSION

We have built an effective simulation tool SimDedup for experimenting with different solutions for data deduplication in primary storage systems. While this study lays the groundwork for SimDedup, we hope to extend it to include many of the aforementioned optimizations in the future. For now, we have demonstrated that SimDedup is able to simulate the process of deduplication with minimal hardware requirements. We have tested SimDedup with multiple different VM traces to validate that it works correctly. SimDedup is easy to use, extensible and free to use for everyone, thus making it the first tool of its kind. We believe that this will be handy for researchers in the field of data deduplication in cloud environments.

REFERENCES

[1] Roger E Bohn and James E Short. 2009. *How Much Information?: 2009 Report on American Consumers.* University of California, San Diego, Global Information Industry Center.

[2] Zhichao Cao, Hao Wen, Xiongzi Ge, Jingwei Ma, Jim Diehl, and David HC Du. 2019. TDDFS: A tier-aware data deduplication-based file system. *ACM Transactions on Storage (TOS)* 15, 1 (2019), 1–26.

[3] Austin T Clements, Irfan Ahmad, Murali Vilayannur, Jinyuan Li, et al. 2009. Decentralized Deduplication in SAN Cluster File Systems.. In *USENIX annual technical conference*. 101–114.

[4] Bo Hong, Demyn Plantenberg, Darrell DE Long, and Miriam Sivan-Zimet. 2004. Duplicate Data Elimination in a SAN File System.. In *MSST*. 301–314.

[5] Jürgen Kaiser, Dirk Meister, André Brinkmann, and Sascha Effert. 2012. Design of an exact data deduplication cluster. In *012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 1–12.

[6] Ricardo Koller and Raju Rangaswami. 2010. I/O deduplication: Utilizing content similarity to improve I/O performance. *ACM Transactions on Storage (TOS)* 6, 3 (2010), 1–26.

[7] Yan-Kit Li, Min Xu, Chun-Ho Ng, and Patrick PC Lee. 2014. Efficient hybrid inline and out-of-line deduplication for backup storage. *ACM Transactions on Storage (TOS)* 11, 1 (2014), 1–21.

[8] Jingwei Ma, Rebecca J Stones, Yuxiang Ma, Jingui Wang, Junjie Ren, Gang Wang, and Xiaoguang Liu. 2017. Lazy exact deduplication. *ACM Transactions on Storage (TOS)* 13, 2 (2017), 1–26.

[9] Sonam Mandal, Geoff Kuenning, Dongju Ok, Varun Shastry, Philip Shilane, Sun Zhen, Vasily Tarasov, and Erez Zadok. 2016. Using hints to improve inline block-layer deduplication. In *14th {USENIX} Conference on File and Storage Technologies ({FAST} 16)*. 315–322.

[10] Dirk Meister and Andre Brinkmann. 2010. dedupv1: Improving deduplication throughput using solid state drives (SSD). In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 1–6.

[11] Dutch T Meyer and William J Bolosky. 2012. A study of practical deduplication. *ACM Transactions on Storage (ToS)* 7, 4 (2012), 1–20.

[12] João Paulo and José Pereira. 2016. Efficient deduplication in a distributed primary storage infrastructure. *ACM Transactions on Storage (TOS)* 12, 4 (2016), 1–35.

[13] Kiran Srinivasan, Timothy Bisson, Garth R Goodson, and Kaladhar Voruganti. 2012. iDedup: latency-aware, inline data deduplication for primary storage.. In *Fast*, Vol. 12. 1–14.

[14] V Tarasov, D Jain, G Kuenning, S Mandal, K Palanisami, P Shilane, and S Trehan. [n.d.]. Dmdedup: Device-mapper deduplication target. In *Proceedings of the Linux Symposium*. 83–95.

[15] Avani Wildani, Ethan L Miller, and Ohad Rodeh. 2013. Hands: A heuristically arranged non-backup in-line deduplication system. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 446–457.

[16] Huijun Wu, Chen Wang, Yinjin Fu, Sherif Sakr, Liming Zhu, and Kai Lu. 2017. Hpdedup: A hybrid prioritized data deduplication mechanism for primary storage in the cloud. *arXiv preprint arXiv:1702.08153* (2017).

[17] Wen Xia, Hong Jiang, Dan Feng, and Yu Hua. 2011. SiLo: A Similarity-Locality based Near-Exact Deduplication Scheme with Low RAM Overhead and High Throughput.. In *USENIX annual technical conference*. 26–30.