# DEDISbench: A Benchmark for Deduplicated Storage Systems

J. Paulo P. Reis J. Pereira A. Sousa

High-Assurance Software Lab (HASLab)
INESC TEC & University of Minho

**Abstract**

Deduplication is widely accepted as an effective technique for eliminating duplicated data in backup and archival systems. Nowadays, deduplication is also becoming appealing in cloud computing, where large-scale virtualized storage infrastructures hold huge data volumes with a significant share of duplicated content. There have thus been several proposals for embedding deduplication in storage appliances and file systems, providing different performance trade-offs while targeting both user and application data, as well as virtual machine images.

It is however hard to determine to what extent is deduplication useful in a particular setting and what technique will provide the best results. In fact, existing disk I/O micro-benchmarks are not designed for evaluating deduplication systems, following simplistic approaches for generating data written that lead to unrealistic amounts of duplicates.

We address this with DEDISbench, a novel micro-benchmark for evaluating disk I/O performance of block based deduplication systems. As the main contribution, we introduce the generation of a realistic duplicate distribution based on real datasets. Moreover, DEDISbench also allows simulating access hotspots and different load intensities for I/O operations. The usefulness of DEDISbench is shown by comparing it with Bonnie++ and IOzone open-source disk I/O micro-benchmarks on assessing two open-source deduplication systems, Opendedup and Lessfs, using Ext4 as a baseline. As a secondary contribution, our results lead to novel insight on the performance of these file systems.

## 1 Introduction

Deduplication is now accepted as an effective technique for eliminating duplicated data in backup and archival storage systems [17] and storage appliances [20], allowing not only to reduce the costs of storage infrastructures but also to have a positive performance impact throughout the storage management stack, namely, in cache efficiency and network bandwidth consumption [13, 12, 10]. With the cloud computing paradigm, applying deduplication to large scale virtualized infrastructures is an emerging trend. In fact, recent studies show that up to 80% of space can be reclaimed for virtual machines with general purpose data [7, 4] and up to 95% for system images in a typical cloud provider [8]. Most strikingly, our previous research shows that approximately 16% of space savings can be achieved even within the confined possibilities of the working files of a research group [16]. These studies also show that these high space saving rates can still be achieved while providing security and privacy for client's data [13].

Deduplication in cloud computing infrastructures raises new challenges for I/O performance and data consistency that are not addressed in backup deduplication. Some cloud applications will access and modify stored data, which we refer as active storage data, in a frequent basis and with strict disk I/O performance requirements, which is not assumed for backup storage data. Backup storage data is immutable and consequently, shared data will never be modified, thus discarding the need of copy-on-write mechanisms to ensure that a storage region is not rewritten while being shared by several entities, which would lead to data corruption. However, copy-on-write mechanisms deteriorate the performance of disk I/O operations by including additional computation in the requests. Yet another challenge arises when deduplication is deployed on a decentralized infrastructure and performed among remote nodes, requiring distributed metadata for indexing the stored content and finding duplicated information. Most deduplication systems use in-band, also know as inline deduplication, where disk I/O write requests to the storage are intercepted and shared before actually being written to the storage. This way, only non-duplicated data is actually written, saving additional storage space but including metadata look up operations inside the critical I/O path, thus increasing I/O latency.

Having in mind these performance challenges and the recent sudden growth of work in this area, it is necessary to have proper benchmarking tools, with realistic workloads, for evaluating the performance of disk I/O when using deduplication systems. Such tools are also necessary for backup deduplication systems despite the distinct requirements. Previous work analyzed 120 datasets used in deduplication studies and concluded that these datasets cannot be used for comparing different systems [18]. Most disk I/O benchmarks do not use a realistic distribution for generating data patterns and, in most cases, the patterns written either have the same content, for each write operations, or have random patterns [5, 9, 3]. If the same content is written for each operation, the deduplication engine will be overloaded with share operations, which will affect the overall performance. Moreover, if this content is rewritten frequently, the amount of copy-on-write operations will increase considerably the I/O operations latency. On the other hand, writing always random content will generate few duplicates and the deduplication systems will be evaluated under a minimal sharing load. Note that generating an unrealistic content workload will affect the disk I/O performance and also the space savings, sharing throughput and resource usage of the deduplication system [18]. Some benchmarks address this in content generation by defining a percentage of duplicated content over the written records [14] or the entropy of generated content [2]. However, these methods are only able to generate simplistic distributions that are not as realistic as desired, or present still preliminary work where several details and proper implementation and evaluation are still missing [18].

We present DEDISbench, a synthetic disk I/O micro-benchmark suited for block based deduplication systems that introduces the following contributions:

- Generation of realistic content distributions, specified as an input file, that can be extracted from real datasets with DEDISgen, an analysis tool also presented at this paper.
- Introduction of an hotspot random distribution, based on TPC-C NURand function [19, 16], that generates access hotspots regions for disk I/O operations.
- I/O operations, for each test, can be performed at a stress load, *i.e.* the maximum load handled by the test machine, or at a nominal load, *i.e.* the throughput of I/O operations is limited to a certain value.

Note that DEDISbench simulates low-level I/O operations and does not focus on generating realistic directory trees and files like other benchmarks [6, 2, 18, 3, 9]. Nevertheless, such benchmarks are also referred along this paper and compared with DEDISbench in terms of content generation and accesses patterns. DEDISbench is evaluated and compared directly with Bonnie++ and IOzone, the two open-source micro-benchmarks that most resemble DEDISbench in terms of the suite and aim of disk I/O tests.

Section 2 presents DEDISbench design, implementation and features, including the novel content and access pattern distributions. Section 3 compares the content and access pattern distributions generated by DEDISbench, Bonnie++ and IOzone. Additionally, this section evaluates Opendedup [15] and LessFS [11] deduplication systems with these three benchmarks and compares their performance with Ext4, a file system without deduplication. Section 4 introduces relevant work and their main differences from DEDISbench. Finally, Section 5 concludes the paper and points DEDISbench main contributions.

## 2 DEDISbench

This section starts by presenting a global overview of DEDISbench design and implementation and then, the generation of realistic content and access pattern distributions are described in more detail.

### 2.1 Basic Design and Features

The basic design and features of DEDISbench resemble the ones found in Bonnie++ [5] and IOzone [14] that are two open-source synthetic micro-benchmarks

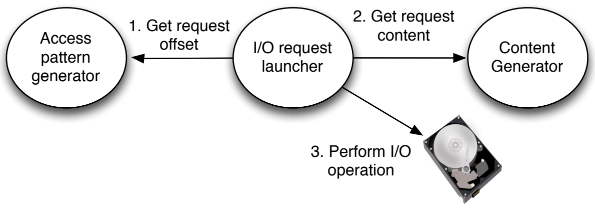

Figure 1: Overview of I/O request generation.

widely used to evaluate disk I/O performance. DEDISbench is implemented in C and allows performing either read or write block disk I/O tests, where the block size is defined by the user. I/O operations can be executed concurrently by several processes with independent files, being the number of processes and the size of process files pre-defined by the users. Moreover, the benchmark can be configured to stop the evaluation when a certain amount of data has been written or when a pre-defined period of time has elapsed, which is not common in most I/O benchmarks. Yet another novel feature of DEDISbench is the possibility of performing I/O operations with different load intensities. In addition to a stress load where the benchmark issues I/O operations as fast as possible to stress the system, DEDISbench supports performing the operations at a nominal load, specified by the user, thus evaluating the system with a stable load. Few I/O benchmarks support both features, as stated in Section 4.

Figure 1 overviews DEDISbench architecture. For each process, an independent I/O request launcher module launches either read or write I/O block operations, at nominal or peak rates, until the termination condition is reached. For each I/O operation, this module must contact the access pattern generator for obtaining the disk offset for the I/O operation (1) that will depend on the type of access pattern chosen by the user and can be sequential, random uniform or random with hotspots. Next, the I/O request launcher module contacts the content generator module for obtaining an identifier for the content to generate (2). Since DEDISbench is aimed at block-based deduplication, this identifier will then be appended as an unique pattern to the block's content, ensuring that blocks with different identifiers will not be duplicated. The generated identifiers will follow the input file provided for DEDISbench with the information about duplicates distribution. Note that this step is only necessary for write I/O requests because read requests do not generate any content to be written. Finally, the operation will be sent to the storage (3) and the metrics regarding operations throughput and latency will be monitored in the I/O request launcher module. Both content and access patterns generation are further detailed next.

## 2.2 I/O Accesses Distribution

DEDISbench can generate sequential and random uniform access patterns for the disk addresses accessed by I/O operations, as in IOzone and Bonnie++. These

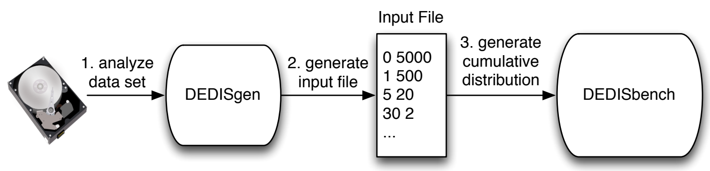

Figure 2: Generating and processing input content distribution file in DEDISbench.

patterns are important to measure the performance of disk arm movement when addresses are accessed sequentially, minimizing the movements, or when the accesses are random, maximizing the arm movement. DEDISbench introduces a novel third access pattern that simulates access hotspots, where few blocks are accessed frequently while the majority of blocks are accessed sporadically. This hotspot distribution is generated with TPC-C NURand function [19, 16]. TPC-C is an industry standard on-line transaction processing SQL benchmark that mimics a wholesale supplier with a number of geographically distributed sales districts and associated warehouses. More specifically, the NURand function is used for generating the addresses to be written in each operation and, as we show in Section 3, this allow us achieving a more realistic pattern, for most applications, that can be used to uncover distinct performance issues.

## 2.3 Duplicates Distribution

DEDISbench main contribution is the ability to process, as input, a file that specifies a distribution of duplicated content, which can be extracted from a real dataset, and using this information for generating a synthetic workload that follows such distribution. As depicted in Figure 2 the input file states the number of blocks for a certain amount of duplicates. In this example there are 5000 blocks with 0 duplicates, 500 blocks with 1 duplicate, 20 blocks with 5 duplicates and 2 blocks with 30 duplicates. This file can be populated by the users or can be generated automatically with DEDISgen, an analysis tool that can be used for processing a real dataset and extracting from it the duplicates information. DEDISgen is implemented in C and processes data from a storage device or from files inside a specific directory tree in the following way: Data from files or from storage devices is read and divided into fixed size blocks, with a size chosen by the user. A SHA-1 hash sum is calculated for each block and inserted in Berkeley DB[1] in order to find duplicated hashes. After processing all data, the database information is transformed into an input file suitable for DEDISbench. DEDISbench uses a default input file based on the content distribution found on our research group storage [16], further explained in Section 3. DEDISbench then uses the input file for generating a cumulative distribution with the probability of choosing a certain block identifier, where two blocks with the same identifier are duplicated. Then with the aid of a random generator, a cumulative function is used for calculating, for each I/O operation, an

[1]Berkeley DB is used as an hash table.

identifier and consequently the content to be written.

## 3 Evaluation

This section compares DEDISbench with IOzone and Bonnie++, which are the two micro-benchmarks with the closest design and features.

Bonnie++ [5] is a standard I/O micro-benchmark that performs several tests to evaluate disk I/O performance *in the following order*: Write tests assess the performance of single byte writes, block writes and rewrites while read tests assess byte and block reads, all with a sequential access distribution. Seek tests perform random block reads and, in 10% of the operations, block writes by following an uniform random distribution. The size of blocks, the number of concurrent Bonnie++ processes and the size of the file each process accesses are defined by the user. All these tests are performed with a stress load and run until an amount of data is written/read for each test. However, it is not possible to specify the content of written blocks. Bonnie++ also tests the creation and deletion of files, which is not contemplated in this evaluation because it is not supported by IOzone or DEDISbench.

IOzone [14] is the I/O micro-benchmark that most resembles DEDISbench and allows performing sequential and random uniform write and read tests. The block size, number of concurrent processes and the size of the files of each process, are also defined by the users. Tests are performed at a stress load and, for each test, the user defines the amount of data to be written by each process. Unlike in Bonnie++ it is possible to define full random tests that perform either read or write random disk I/O operations. Additionally, the percentage of duplicated inter-file and intra-file content in each block can also be specified. However, as discussed in the next sections, this content generation mechanism does not allow specifying a content distribution with a realistic level of detail as in DEDISbench.

DEDISbench, IOzone and Bonnie++ have several features in common but also differ in specific details that are discussed and evaluated thorough this section and that influence the evaluation of deduplication systems. The remaining of this section compares the content and disk access patterns generated by each benchmark and points the main differences from DEDISbench. Then, two open-source deduplication file systems, Opendedup and Lessfs, are evaluated with the three benchmarks and compared with Ext4, a standard file system without deduplication, thus allowing us to analyze how the different content and accesses distributions, used by each benchmark, may influence the evaluation results.

### 3.1 Evaluation setup

All tests ran in a 3.1 GHz Dual-Core Intel Core Processor with hyper-threading, 4GB of RAM and a local SATA disk with 7200 RPMs. Unless stated otherwise, in all tests the total amount of data written was 8GB distributed over 4 concurrent processes, each writing 2GB in an independent file. The block size chosen was 4KB for DEDISbench and Bonnie++, except in Bonnie++ single byte tests. For IOzone, the block size chosen was 16KB in order to use the content generator and being able to specify that 4KB of the full block were duplicated, thus simulating a percentage of duplicated data that resembled the one used by DEDISbench. As explained previously, DEDISbench uses a default input content distribution that was extracted from the research data of our group [16]. Briefly, this dataset has approximately 1.5 million personal files from our research projects that consume approximately 110GB of storage space. DEDISgen was used to analyze this real dataset, with a block size of 4KB, and to generate a custom input distribution for DEDISbench. In this dataset, approximately 76% of the blocks do not have any duplicate, while 18% of the blocks are duplicated and can be eliminated. The remaining 6% belongs to unique blocks that have duplicated content. This is why IOzone block size was chosen to be 16KB, allowing us to define that each block would have 25% of its data (4KB) duplicated among distinct process files. With this configuration and using 4 independent files, each block of 16KB as a distinct 4KB region with three duplicates, one for each file, which resembles the average number of 2.76 duplicates per block found in our research group dataset and generates 18.5% of duplicated blocks that can be eliminated. The remaining 75% of the blocks are not duplicated.

IOzone allows defining both intra and inter-file duplicates, for example, it would be possible to define that a block region is only duplicated in the same file, which would generate few regions with several duplicates in our experimental setup. Nevertheless, this would increase greatly the block size and the complexity of the configuration to achieve a similar distribution to the real dataset and, even with these modifications, the level of detail would still be limited when compared to DEDISbench. In IOzone one could simulate two or three distinct types of blocks with a distinct proportion of duplicates while in DEDISbench it is possible to simulate as many types as specified in the input distribution file.

## 3.2 Duplicates Distribution

Before running the benchmarks on deduplication systems, we analyzed the content generated by each benchmark. The results discussed in this section were extracted with DEDISgen that processed the files generated by each benchmark after completing a sequential disk I/O write test. We choose the sequential I/O test over a random test because there are no block rewrites, enabling the extraction of precise information about all the written blocks and their contents.

Figure 3 presents the percentage of unique blocks with a certain range of duplicates (*i.e.* equal to 0, between 1 and 5, 5 and 10, 10 and 50 and so on) for Bonnie++, IOzone, DEDISbench and the distribution extracted from our research group dataset. All the distinct blocks generated by Bonnie++ have between 1 and 5 duplicates, in fact, each unique block has precisely 3 duplicates because every file is written with the exact same content, meaning that, all blocks in the same file are distinct but are duplicated among the other files. Consequently, as shown in Table 1, with Bonnie++ 75% of the written space can be deduplicated which may introduce a significant load in the deduplication, copy-on-write and garbage collection engines. Note that, Figure 3 shows the number of duplicates generated for each *unique block* written by the benchmarks while Table 1 shows the percentage of unique blocks without any duplicate, with duplicates and the percentage of duplicated blocks for *all the blocks* written by the benchmarks, thus explaining why the percentages differ.

The results for IOzone in Figure 3 show that most unique blocks do not have any duplicate, while the remaining blocks have mainly between 1 and 5 duplicates and a very small percentage has between 5 and 10. In fact, the remaining distinct blocks should have 3 duplicates each, which happens for almost all the blocks with the exception of 216 blocks that have only 1 duplicate and 3 blocks that have 7 duplicates. If the content of unique blocks is generated randomly, it is possible to have these collisions, which are not significant for the number of 4KB blocks found in the 8GB written by the I/O tests. In Table 1, IOzone percentage of duplicates and unique blocks is closer to the percentages found at the real distribution.

The results of DEDISbench, in Figure 3, show that the number of unique blocks and their duplicates is distributed over several regions, which is more realistic when compared to the real distribution. DEDISbench generates most blocks with few duplicates and a small percentage of blocks with many duplicates. In fact, we omitted one value from the figure in the far end of the distribution tail, for legibility reasons, where a single block has 15665 duplicates. As depicted in this figure, DEDISbench distribution is much closer to a real dataset which may impact the performance of deduplication systems. For example, having many blocks with few duplicates will increase the number of shared blocks that, after being rewritten, must be collected by the garbage collection algorithm. On the other hand, mixing blocks with different number of duplicates will also affect the size of metadata structures and the work performed by the deduplication engine. However, when looking at Table 1 the results are slightly more distant from the real values when compared to IOzone results. This happens due to the algorithm used by DEDISbench to generate the cumulative content distribution and due to the dataset where the distribution input information was extracted from. The size of the real dataset is above 100GB while the benchmark is only writing 8GB, meaning that many of the duplicated blocks are being written only once, even if the cumulative distribution has a high probability for writing these blocks. Figure 4 and Table 2 compare the results of running DEDISbench sequential write tests for 16 and 32GB (divided by 4 files) and shows that when the amount of written data is closest to the amount of data in the real dataset, the distribution generated by DEDISbench also becomes closer to the real one.

To sum up, these results show that both Bonnie++ and IOzone do not simulate accurately the distribution of duplicates per unique blocks. This detail can influence the load in the deduplication and garbage collection mechanisms of the deduplication system. For instance, a block shared by two entities or by one hundred

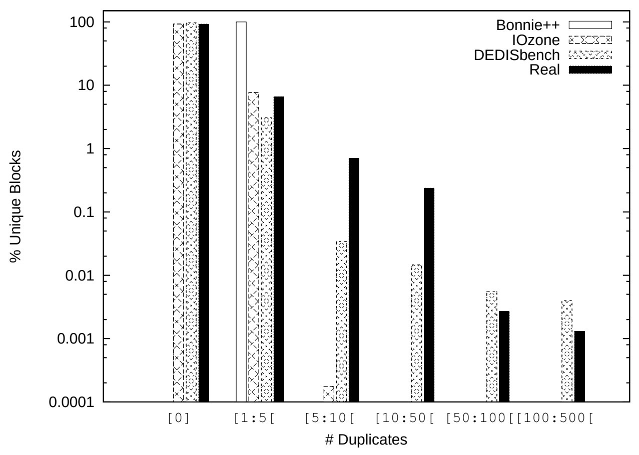

Figure 3: Distribution of duplicates ranges per distinct blocks for Bonnie++, IOzone, DEDISbench and the real dataset.

Table 1: Unique and Duplicated blocks in Bonnie++, IOzone,DEDISbench and the real dataset.

| | Bonnie++ | IOzone | DEDISbench | Real |
|---|---|---|---|---|
| % Unique blocks with 0 duplicates | 0 | 75 | 90 | 76 |
| % Unique blocks with duplicates | 25 | 6 | 3 | 6 |
| % Duplicated blocks | 75 | 19 | 7 | 18 |

determines the timing when garbage collection is needed, how often the copy-on-write mechanism must be used and the amount of information in metadata structures for sharing identical content. In Bonnie++ sharing an excessive amount of blocks will overload the deduplication engines while in IOzone, having all duplicated blocks with 3 or 4 distinct duplicates may also not be realistic and influence the evaluation.

## 3.3 I/O Accesses Distribution

Another contribution of DEDISbench is the introduction of the NURand hotspot distribution besides the traditional sequential and random uniform disk access patterns, used in Bonnie++ and IOzone. We ran DEDISbench with the three access distributions: sequential, random uniform and NURand, and extracted the access patterns of each distribution. Only DEDISbench was used in these tests because extracting this information from IOzone and Bonnie++ would require modifying their source code. Moreover, DEDISbench sequential and random uniform distributions mimic the ones found in these two benchmarks.

Figure 5 presents the percentage of blocks for a certain range of accesses. In

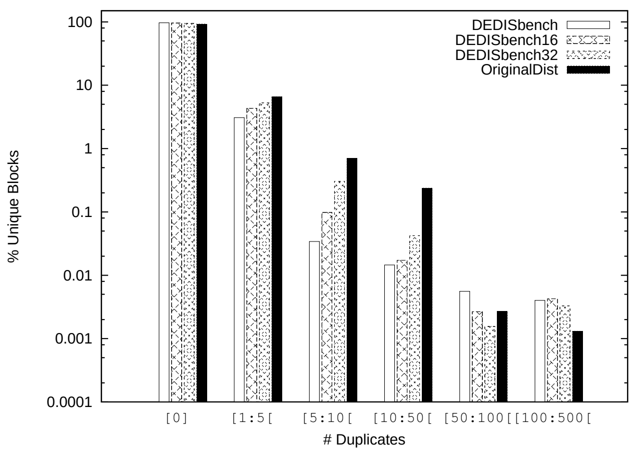

Figure 4: Distribution of duplicates ranges per distinct blocks for DEDISbench tests of 8,16 and 32 GB and for the Real dataset.

Table 2: Unique and duplicated blocks in DEDISbench datasets with 8,16 and 32 GB and in the Real dataset.

| | DEDISbench 8 | DEDISbench 16 | DEDISbench 32 | Real |
|---|---|---|---|---|
| % Unique blocks with 0 duplicates | 90 | 87 | 83 | 76 |
| % Unique blocks with duplicates | 3 | 4 | 5 | 6 |
| % Duplicated blocks | 7 | 8 | 12 | 18 |

the sequential distribution 100% of the blocks are accessed precisely once (range between 1 and 5 in the figure) while in the random uniform distribution most of the blocks are accessed between 1 and 5 times, in fact most blocks are accessed only once and the percentage of blocks decreases with the number of accesses. On the other hand, the NURand distribution shows that a high percentage of blocks is accessed few times while a small percentage is accessed many times, generating blocks that are hotspots (*i.e.* a few blocks are accessed more than 500 times).

These results show that, with the NURand distribution, it is possible to create hotspots for I/O requests, thus generating blocks that are constantly being accessed, which for deduplication systems means blocks that are constantly being shared, copied-on-write and garbage collected. Such environment may be more appropriate for evaluating deduplication systems where some data is accessed frequently and most data is only accessed sporadically, which is not contemplated in sequential and random uniform distributions.

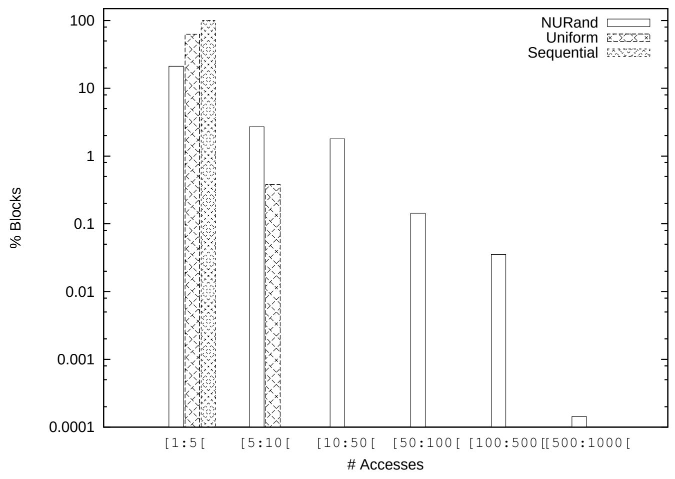

Figure 5: Distribution of accesses per block for Sequential, Random Uniform and NURand approaches.

## 3.4 I/O Performance Evaluation

After comparing the content and access distributions of DEDISbench, IOzone and Bonnie++, we have analyzed how these benchmarks evaluate two distinct deduplication systems.

LessFS [11] is an open source single-host deduplication file-system designed mainly for backup purposes but that can also be used for storing VM images and active data with moderate performance. LessFS uses FUSE for implementing file-system semantics and only supports in-line deduplication. Data is stored as chunks with a fixed size of 4KB.

Opendedup [15] is an open-source deduplication file system that supports single and multi-host deduplication. Deduplication can be performed in-line or in batch (off-band) mode, where data is shared in an asynchronous fashion only after being stored. Opendedup file-system is based on FUSE to export common file system primitives and uses Java for the deduplication engine. Data is stored as chunks and the block size can be parametrizable.

These two systems were chosen because they are open-source systems in a mature development and export file systems supporting data modification, unlike in most backup deduplication systems, which is needed for testing the impact of copy-on-write and garbage collection mechanisms. The three benchmarks also ran on Ext4, a traditional file system without deduplication. All the file systems were mounted in the same partition, with a size of 20GB, that was formatted before running each benchmark. Also, all the deduplication file systems were configured to have a block size of 4KB and perform deduplication in-line. By performing in-line deduplication, data is shared immediately and the consequent overheads are also visible immediately, which would not be possible in batch mode deduplication.

Table 3: Evaluation of Ext4, LessFS and Opendedup with Bonnie++.

| | Ext4 | Lessfs | Opendedup |
|---|---|---|---|
| Sequential byte write (KB/s) | 1100 | 76 | 56 |
| Sequential block write (KB/s) | 72035 | 13860 | 155496 |
| Sequential block rewrite (KB/s) | 17319 | 1016 | 62744 |
| Sequential byte read (KB/s) | 3029 | 1262 | 72 |
| Sequential block read (KB/s) | 73952 | 60064 | 144614 |
| Urandom seek (KB/s) | 170.9 | 127.1 | 115.8 |

Table 4: CPU and RAM consumption of LessFS and Opendedup for Bonnie++, IOzone and DEDISbench.

| | | LessFS | Opendedup |
|---|---|---|---|
| Bonnie++ | CPU | 22 % | 163 % |
| | RAM | 2.2 GB | 1.8 GB |
| IOzone | CPU | 9 % | 25 % |
| | RAM | 1.25 GB | 2.1 GB |
| DEDISbench | CPU | 15.7 | 19.5 % |
| | RAM | 2.2 GB | 1.9 GB |

In Bonnie++ the tests were performed in the following order, which cannot be changed: single-byte write, block write and block rewrite in sequential mode, single-byte read and block read in sequential mode and, finally, the random seek test. For IOzone and DEDISbench we choose the tests order to be as similar as possible to Bonnie++. In IOzone the order was: Block write, block rewrite, block read and block reread in sequential mode and block read and block re-read in random uniform mode. For DEDISbench the order was exactly the same as in IOzone but we introduced two more tests before ending the benchmark that were the block read and block write with the NURand distribution. Finally, we do not compare directly the results of different benchmarks since each benchmark has different implementations for calculating throughput rates. Instead, we analyze each benchmark independently and compare the overhead of Opendedup and LessFS deduplication filesystems over Ext4 fylesystem that does not perform any deduplication.

Table 3 shows the results of running Bonnie++ on Ext4, LessFS and Opendedup. By comparing the deduplication systems with Ext4 it is possible to conclude that writing sequentially one byte at a time is inefficient because, for each written byte, a block of 4KB will be modified and will be shared by the deduplication system, thus forcing the deduplication system to process a single block 4096 times. This is also true for sequential byte reads where, in each operation, it must be made an access to the metadata that tracks the stored blocks for retrieving a single byte. In this last test, the overhead introduced by Opendedup, when compared to LessFS overhead, is considerably higher and can be caused by retrieving the whole block to memory in each byte read operation instead of taking advantage of

Table 5: Evaluation of Ext4, LessFS and Opendedup with IOzone.

| | Ext4 | Lessfs | Opendedup |
|---|---|---|---|
| Sequential block write (KB/s) | 74463 | 5525,24 | 19760,8 |
| Sequential block rewrite (KB/s) | 74356,88 | 373,28 | 29924,84 |
| Sequential block read (KB/s) | 67159,36 | 7777,48 | 10464,4 |
| Sequential block reread (KB/s) | 67522,48 | 11495,48 | 10403,72 |
| Urandom block read (KB/s) | 2086,4 | 1304,08 | 1766,24 |
| Urandom block write (KB/s) | 2564,76 | 162,4 | 1608,04 |

a caching mechanism. In sequential block write and rewrite Opendedup outperforms Ext4 by taking advantage of Bonnie++ writing the same content in all tests. Data written in sequential byte tests was already shared and Opendedup algorithm only requires consulting the in-memory metadata for finding duplicated content and sharing it, thus avoiding the need of actually writing the new blocks to disk. On the other hand, LessFS implementation does not seem to take advantage of such scenario. Opendedup also outperforms Ext4 in sequential block reads probably with a cache mechanism, efficient only at the block granularity. Finally, in random seek tests both deduplication systems present worse results that Ext4, with LessFS slightly outperforming Opendedup. RAM and CPU usages while Bonnie++ was running are depicted in Table 4. Both Opendedup and LessFS consume a significant amount of RAM, meaning that most metadata is loaded in memory and explaining, for example, the performance boosts of Opendedup in sequential block read and write tests. Moreover, the increase in CPU consumption with Opendedup can be a consequence of Bonnie++ writing a high percentage of duplicated content, thus generating an unrealistic amount of duplicated data to be processed.

Table 5 shows the results of running IOzone on Ext4, LessFS and Opendedup. Unlike Bonnie++, this benchmark does not write always the same content explaining why Opendedup does not outperforms Ext4 in block rewrite operations. Although some of the data was shared already, the content written is not always the same and most requests are still written to disk. With IOzone, Opendedup outperforms LessFS in almost all tests with the exception of block re-read test where LessFS is slightly better. LessFS decrease in performance is more visible in sequential and random write tests and mainly in re-write tests. In Table 4, the RAM and CPU usages drop significantly which can be a consequence of writing less duplicated content. The RAM usage in Opendedup is an exception and the value is higher than in Bonnie++ tests.

Table 6 shows the results of running DEDISbench on Ext4, LessFS and Opendedup. As explained previously, IOzone generates all duplicated blocks with exactly 3 duplicates while DEDISbench uses a realistic distribution where most blocks have few duplicates but some blocks are highly duplicated, which will help explaining the next results. In sequential tests both Opendedup and LessFS are outperformed by Ext4, as in IOzone evaluation. However, the results of Opendedup

Table 6: Evaluation of Ext4, LessFS and Opendedup with DEDISbench.

| | Ext4 | Lessfs | Opendedup |
|---|---|---|---|
| Sequential block write (KB/s) | 86916.82 | 5025.352 | 77508.424 |
| Sequential block rewrite (KB/s) | 76905.028 | 658.324 | 18852.732 |
| Sequential block read (KB/s) | 78648.964 | 7527.196 | 18591.672 |
| Sequential block reread (KB/s) | 78620.46 | 11788.792 | 20404.88 |
| Urandom block read (KB/s) | 791.356 | 2055.228 | 511.62 |
| Urandom block write (KB/s) | 1416.016 | 123,232 | n.a. |
| NURandom block read (KB/s) | 2287.208 | 1829.704 | 1350,304 |
| NURandom block write (KB/s) | 1246.336 | 151.556 | n.a. |

for the sequential write test show considerably less overhead when compared to the same IOzone test, which can be a consequence of DEDISbench generating some blocks with a large amount of duplicates that will require writing only one copy to the storage, thus enhancing the performance of Opendedup. On the other hand in the sequential rewrite tests, Opendedup performance decreases since DEDISbench generates many blocks with few duplicates that will then be rewritten and will require garbage collection, thus increasing the overhead.

The most interesting results appear in the random I/O tests. Firstly, LessFS outperforms Ext4 in uniform random block read test, which is an harsh test for the disk arm movement, pointing one of the advantages of using deduplication. If two blocks stored in distant disk positions are shared, the shared block will then point to the same disk offset and a disk arm movement will be spared. In IOzone there are few duplicates per block and this operations does not occur so often but, in DEDISbench some blocks have a large number of duplicates which can reduce significantly the disk arm movement and consequently improve performance. Even in Opendedup where this improvement is less visible, the overhead for random uniform read tests is lower than the one for sequential read tests. With the NURand hotspot distribution the performance of read operations in Ext4 is leveraged because caching mechanisms can be used more efficiently, thus the performance LessFS and Opendedup are reduced but, nevertheless, achieve better performance than in sequential tests. The CPU and RAM consumptions, shown in Table 4, for LessFS and Opendedup are similar to the ones obtained with IOzone, with a slight reduction in Opendedup and increase in LessFS. These variations can be explained by the design and implementation of each deduplication system and how these process the distinct generated datasets.

The other interesting results are visible in the uniform and NURand random write tests. The performance of LessFS when compared to Ext4 decreases significantly while Opendedup system blocks with a CPU usage of almost 400%, not being able to complete these tests. Realistic content distribution in DEDISbench uncovered a problem in Opendedup that could not be detected with simplistic content distributions in IOzone and Bonnie++. To further prove this point, Table 7

Table 7: Evaluation of Opendedup with DEDISbench and a modified version of DEDISbench that generates the same content for each written block.

| | DEDISbench Original | DEDISbench Modified |
|---|---|---|
| Sequential block write (KB/s) | 77508.424 | 247428,092 |
| Sequential block rewrite (KB/s) | 18852.732 | 253817,508 |
| Sequential block read (KB/s) | 18591.672 | 412694,064 |
| Sequential block reread (KB/s) | 20404.88 | 418169,436 |
| Urandom block read (KB/s) | 511.62 | 106696,336 |
| Urandom block write (KB/s) | n.a. | 3638,368 |
| NURandom block read (KB/s) | 1350,304 | 73385,616 |
| NURandom block write (KB/s) | n.a. | 3288,78 |
| % CPU consumption | 19 | 272 |
| RAM consumtion (GB) | 2.1 | 2.6 |

tests Opendedup with the default DEDISbench and a modified version that writes always the same content, in each I/O operation, and, as we can see by the results, Opendedup completes successfully all the tests and greatly increases the performance, even when compared to the Ext4 results with the default DEDISbench version. However, the drawback of processing a fully duplicated dataset is visible in the CPU and RAM usage of Opendedup that increase to 272% and 2.6 GB respectively, which can be a serious limitation for deduplication in cloud commodity servers. Furthermore, these results show that using a realistic content distribution is necessary for a proper evaluation of deduplication systems and that Opendedup is not thought for datasets with a higher percentage of non-duplicated data.

This section states that using realistic content and accesses distributions influences significantly the evaluation of deduplication systems. Moreover, generating a realistic content distribution is necessary for finding performance issues and system design fails, like the ones found in Opendedup, but also for finding deduplication advantages, such as the boost in performance of uniform random read tests in LessFS. Moreover, it is useful having a benchmark that can simulate several content distributions ranging from fully duplicated to fully unique content and, most importantly, that is able to generate a content distribution where the number of duplicates per block is variable and follows a realistic distribution. To our knowledge, this is only achievable with DEDISbench.

## 4 Related Work

Despite the extensive research on I/O benchmarking, to our knowledge and as discussed in previous published work, I/O benchmarks that allow defining content distributions are vaguely addressed in the literature and are either limited to generating simplistic distributions [14, 6] or are still preliminary work [18].

A lot like DEDISbench does, IOzone [14] and Bonnie++ [5] test disk I/O performance by performing concurrent sequential and random read and write operations in several files. Bonnie++ does not allow specifying the content generated for I/O operations, in fact, it writes the same content in each disk I/O test and for each file. On the other hand, IOzone allows specifying the percentage of duplicated data in each record (block). It is possible to subdivide further this duplicate percentage and detail the amount of this percentage that is duplicated among other records in the same file (intra-file), among records on distinct files (inter-file) and in both intra and inter file. In other words, these parameters allow defining the percentage of duplicated and non duplicated intra and inter-file content, meaning that, it is possible to achieve some control over the number of duplicates per record and have different regions of a record with a different number of duplicates like in DEDISbench. However, achieving such distributions can be complex and the level of detail will never be as realistic as the one provided by DEDISbench. Both IOzone and Bonnie++ use either sequential or random uniform distributions for the access pattern of I/O operations and are only able to perform stress testing. In Bonnie++ and IOzone, tests are performed at a peak/stress rate and random tests follow an uniform random distribution that balances equally the I/O operations per file region. DEDISbench introduces an hotspot access pattern distribution based on TPC-C NURand function and allows to perform I/O operations at a nominal throughput, that may be more realistic settings for most applications. To our knowledge, Bonnie++ and IOzone are the closest synthetic micro-benchmarks to DEDISbench in terms of design principles and evaluation parameters, which is why we compare our benchmark directly with both in Section 3.

Other work, with different assumptions from DEDISbench, IOzone and Bonnie++, leverages the simulation of actual file systems by generating directory threes and depth, the amount of files in each directory, distinct file sizes and multiple operations on files and directories. Most of these benchmarks use probabilistic distributions for building filesystem trees, choosing the operations to execute and the targets of each operation [3, 1, 9, 6, 18]. Fstress [3] presents several workloads with different distributions (*e.g.* peer-to-peer, mail and news servers) that run with a predefined nominal load like in DEDISbench. Moreover, Fstress also uses an hotspot probabilistic distribution for assigning operations to distinct files. Postmark [9] is designed to evaluate the performance of creating, appending, reading and deleting small files, simulating the characteristic workloads found in mail, news and web-based commerce servers. Target files and sizes are choosen by following an uniform distribution. Agrawal et all [1] work uses distinct probability models for creating new directories and files, for choosing the depth and number of files in each directory and for choosing the size and the access patterns to distinct files. However, none of these benchmarks allows specifying the content to be written, using instead a random or a constant pattern.

Filebench [6, 2] uses an entropy based approach for generating data with distinct content, for each I/O operation, that allows controlling the compression and duplication ratio of a dataset. Like in IOzone, this approach allows simulating the amount of duplicated data in a specific dataset but does not allow detailing further the distribution like in DEDISbench. Furthermore, we could not find the implementation details of this feature in the current version of Filebench. Tarasov *et al.* [18] preliminary work presents a framework for generating data content and metadata evolution in a controllable way. Their algorithm builds a base image with pre-defined directories and files and then uses a Markov model to perform file-level changes and multi-dimensional statistics to perform in-file changes that result in mutations of the base image. Metadata and data changes are loaded from pre-computed profiles. extracted from public and private data of different web servers, e-mail servers and version control repositories. This is still preliminary work, thus lacking details about the generation and loading of the duplicates distribution that is neither detailed nor evaluated. Despite the different aims of DEDISbench and these filesystem benchmarks, our content generation algorithm is still different from the ones found in these systems and could be incorporated, with some design and implementation modifications, in any of these benchmarks.

To sum up, most I/O benchmarks do not support the generation of duplicated content writing either random or constant data patterns. To our knowledge, IOzone, Filebench and Tarasov et all [18] are the only I/O benchmarks that support such feature but, when compared with DEDISbench, these benchmarks use different algorithms for generating duplicated content that are lacking design and implementation details or limiting the realism of the generated distributions.

## 5 Conclusion

This paper presents DEDISbench, a synthetic disk I/O micro-benchmark designed for evaluating deduplication systems. As the main contribution, DEDISbench can process metadata extracted from real datasets and use it to generate realistic content for I/O write operations. Existing I/O benchmarks, either do not focus on distinct content generation or generate limited distributions that, in most cases, do not simulate accurately a real dataset. We also introduce DEDISgen, a tool for analyzing real datasets and extracting the necessary metadata to be used as input by DEDISbench for generating realistic content distributions.

As other contributions, DEDISbench introduces an hotspot distribution, based on TPC-C NURand function, that generates a random access pattern for I/O operations where few blocks are hotspots and the remaining blocks are accessed sporadically. This simulates, for many applications, a more realistic pattern than the traditional random uniform one. Finally, DEDISbench can also perform I/O tests with stress and nominal intensities.

The comparison of DEDISbench with IOzone and Bonnie++ shows that DEDISbench simulates more accurately a real content distribution, allowing to specify in detail the proportion of duplicates per unique block. This increased accuracy was key for finding new performance advantages and drawbacks and also system issues in two deduplication file systems, LessFS and Opendedup, evaluated with the three benchmarks and compared to Ext4, a file system without deduplication. In fact, DEDISbench realistic content distribution uncovered an important limitation in Opendedup implementation. Finally, DEDISbench hotspot access distribution allowed evaluating the performance of random disk accesses, in each system, while maintaining some cache performance, which is not possible with the random uniform distribution, used by Bonnie++ and IOzone.

To conclude, DEDISbench is, to our knowledge, the only disk I/O micro-benchmark that simulates content distributions, extracted from distinct real datasets, with a level of detail that allows evaluating accurately deduplication systems.

## 6 Availability

DEDISbench documentation, source code and debian packages are available at: `http://www.holeycow.org/Home/dedisbench`.

## 7 Acknowledgments

This work is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project RED FCOMP-01-0124-FEDER-010156 and FCT by Ph.D scholarship SFRH-BD-71372-2010.

## References

[1] Nitin Agrawal, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Generating realistic impressions for file-system benchmarking. In *Conference on File and Storage Technologies*, 2009.

[2] Rami Al-Rfou, Nikhil Patwardhan, and Phanindra Bhagavatula. Deduplication and compression benchmarking in filebench. Technical report, 2010.

[3] Darrell Anderson. Fstress: A flexible network file service benchmark. Technical report, 2002.

[4] Austin T. Clements, Irfan Ahmad, Murali Vilayannur, and Jinyuan Li. Decentralized deduplication in san cluster file systems. In *USENIX Annual Technical Conference*, 2009.

[5] Russell Coker. Bonnie++ web page. `http://www.coker.com.au/bonnie++/`. May 2012.

[6] Filebench. Filebench web page. `http://filebench.sourceforge.net`. May 2012.

[7] Gregory R. Ganger and John Wilkes. A study of practical deduplication. In *Conference on File and Storage Technologies*, 2011.

[8] White paper - complete storage and data protection architecture for vmware vsphere. Technical report, 2011.

[9] Jeffrey Katcher. Postmark: a new file system benchmark. Technical report, 1997.

[10] Ricardo Koller and Raju Rangaswami. I/o deduplication: utilizing content similarity to improve i/o performance. In *Conference on File and Storage Technologies*, 2010.

[11] Lessfs. Lessfs web page. `http://www.lessfs.com/wordpress/`. May 2012.

[12] Athicha Muthitacharoen, Benjie Chen, David Mazieres, and David Mazi Eres. A low-bandwidth network file system. In *Symposium on Operating Systems Principles*, 2001.

[13] Partho Nath, Michael A. Kozuch, David R. Ohallaron, Jan Harkes, M. Satyanarayanan, Niraj Tolia, and Matt Toups. Design tradeoffs in applying content addressable storage to enterprise-scale systems based on virtual machines. In *USENIX Annual Technical Conference*, 2006.

[14] William D. Norcott. Iozone web page. `http://www.iozone.org/`. May 2012.

[15] Opendedup. Opendedup web page. `http://opendedup.org`. May 2012.

[16] João Paulo. Efficient storage of data in cloud computing. Master's thesis, 2009.

[17] Sean Quinlan and Sean Dorward. Venti: A new approach to archival storage. In *Conference on File and Storage Technologies*, 2002.

[18] Vasily Tarasov, Amar Mudrankit, Will Buik, Philip Shilane, Geoff Kuenning, and Erez Zadok. Generating realistic datasets for deduplication analysis. In *USENIX Annual Technical Conference. Poster Session*, 2012.

[19] Transaction processing performance council. TPC-C standard specification, revision 5.5. `http://www.tpc.org/tpcc/spec/tpcc_current.pdf`.

[20] Benjamin Zhu, Kai Li, and Hugo Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *Conference on File and Storage Technologies*, 2008.