Masaryk University
Faculty of Informatics

Performance testing of Virtual Data Optimizer storage layer

Master's Thesis

Samuel Petrovič

Brno, Fall 2019

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Samuel Petrovič

Advisor: Adam Rambousek

Acknowledgements

I would like to thank my self for tremendous help and guidance during writing of this thesis. I would also like to thank Red Hat for collaboration and provision of necessary testing equipment.

Abstract

Abstrakt sa pise nakoniec

Keywords

file system, performance, aging, benchmarking, fragmentation, storage, trim, fs-drift, XFS, VDO

Contents

List of Tables

List of Figures

1 Introduction

File systems remain an important part of modern storage solutions. Large, growing databases, multimedia and other storage based applications need to be supported by high-performing infrastructure layer of storing and retrieving information. Such infrastructure have to be provided by the operating systems (OS) in a form of file system.

2 Related work

In this chapter I present different approaches of file system aging and fragmentation research described and implemented in the past. The first section discuss usage of collected data to create aging workload. The second section discuss possibilities of aging the file system artificially, without pre-collected data.

2.1 Aging file system using real-life data

3 File system and storage devices

In this chapter, I present basic information about used file systems, storage devices and its features relevant to performance and aging.

4 Virtual Data Optimiser

4.1 Introduction

Virtual Data Optimizer (VDO) is a compression layer in kernel storage stack. Using block virtualisation, it allows for users to operate with a logic volume much greater than physically available. This sort of overprovision is achieved by using block deduplication an block compression.

Deduplication is a technique that, on a block level, dissalows multiple copies of the same data to be written on physical device. In VDO, duplicate blocks are detected but only one copy is physically written. Subsequent copies then only reference the address of that written block. These blocks are therefore called shared blocks.

Compression is a technique that reduces usage of physical device by identifying and eliminating redundancy in data. In VDO, lossless HIOPS compression, based on a parallelized packaging algorithm is used to handle many compression operations at once. Compressed blocks are stored in a way that allows the most logical blocks to be stored in one physical block.

The actual VDO technology consists of two kernel modules. First module, called `kvdo`, loads into the Linux Device Mapper layer to provide a deduplicated, compressed, and thinly provisioned block storage volume. Second module called `uds` communicates with the Universal Deduplication Service (UDS) index on the volume and analyzes data for duplicates.

4.1.1 Deduplication

Deduplication in VDO relies on growing UDS index. Any incomming block requested to be written is hashed. The hash is then serached for in the UDS index. In case it is found, reference on the written block is retrieved and returned, succesfully deduplicating the incomming block. The hash entry is then moved to the beginning of the index. In case the hash is not found in the index, the entry with hash is written at the beggining of the index and the block is passed to the compression phase. This index is held in memory to present quick deduplication

advice to the VDO volume, therefore there is a minimum requirement of 250 MB of DRAM to be available.

4.1.2 Compression

4.2 VDO Layer

VDO layer is actually another block device that can aggregate physical storage, partitions etc. On creation of VDO volume, management tool also creates volume for UDS index as well as for the actual VDO.

4.2.1 Physical Size

The VDO volume is divided into continuous regions of physical space of constant size. These regions are called slabs and maybe of size of any power of 2 multiple of 128 MB up to 32 GB . After creating VDO volume, the slab size cannont be changed. However, a single VDO volume contain only up to 8096 slabs, so the configured size of slab at VDO volume creation determines its maximum allowed physical storage. Since the maximum slab size is 32 GB and maximum number of slabs is 8096, the maximum volume of physical storage usable by VDO is therefore 256 TB. Important thing to notice is that at least one slab will be reserved for VDO metadata and wouldn't be used for storing data. Slab size does not affect VDO performance.

When trying to examine physical size, two terms are offered. The term Physical size stands for the overall size of undelying device. Available physical size stands for the portion of physical size, that can actually hold user data. The part that does not hold user data is used for storing VDO metadata, f.e. UDS index.

4.2.2 Logical Size

The concept of VDO offers the user means for overprovisioning the physical volume. During creation of VDO volume, user can specify logical size of volume, which can be much larger than the size of physical underlying storage. The user should be able to predict the compressibility of future incoming data and set the logical volume

accordingly. At maximum, VDO suports up to 254 times the size of physical volume which amounts to maximum logical size of 4 PB

4.2.3 Memory requirements

The VDO module itself requires 370 MB and additional 268 MB per every 1 TB of used physical storage. Users are therefore expected to compute the needed memory volume and act accordingly.

Another module that consumes memory is the UDS index. However, several mechanisms are in place to ensure the memory consumption does not offset the advantages of VDO usage.

There are two parts to UDS memory usage. First is a compact representation in RAM that contains at most one entry per unique block, that is used for deduplication advice. Second is stored on-disk that keeps track of all blocks presented to UDS. The part stored in RAM tracks only most recent blocks and is called *deduplicationwindow*. Despite it being only index of recent data, most datasets with large levels of possible deduplication also show a high degree of temporal locality, according to developers. This allows for having only a fraction of the UDS index in memory, while still mantaining high levels of deduplication. Were not for this fact, memory requirements for UDS index would be so high that it would out-cost the advantages of VDO usage completely.

For better memory usage, UDS's Sparse Indexing feature was introduced to the uds module. This feature further exploits the temporal locality quality by holding only the most revelant index entries in the memory. Using this feature (which is reccomended default for VDO) allows for maintaining up to ten times larger deduplication window whlie maintaining the same memory requirements.

4.2.4 Storage requirements

As mentioned earlier, at least one *slab* is reserved for VDO metadata and UDS on-disk index.

VDO module keeps two kinds of metadata which differ in the scale of required space.

1. type scales with physical size and uses about 1 MB per every 4 GB of managed physical storage and also additional 1 MB per *slab*.
2. type scales with logical size and uses approximately 1.25 MB for every 1 GB of logical storage, rounded up to the nearest slab.

4.2.5 VDO in Storage Stack