Masaryk University
Faculty of Informatics

The effects of age on file system performance

Bachelor's Thesis

Samuel Petrovic

Brno, Spring 2017

Replace this page with a copy of the official signed thesis assignment and a copy of the Statement of an Author.

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 Petrovic

Advisor: Adam Rambousek

Acknowledgement

This is the acknowledgement for my thesis, which can span multiple paragraphs.

Abstract

This is the abstract of my thesis, which can span multiple paragraphs.

Keywords

filesystem, xfs, IO operation, aging, fragmentation ...

Contents

List of Tables

List of Figures

1 Introduction

We live in an age of information. Hardware and software technology improves every day. There is a need to work with large databases, multi-media applications and to store great amount of data on a memory device. This cause great pressure on performance of storing and retrieving information, e.g. input and output (I/O) operations.

Part of an operating system that handles communication with a physical device is called a file system. Because of such increasing requirement on performance of I/O operations, there has been quite wide technological progress in approach to performance issues.

In past years, great many tools and tests (e.g. benchmarks) were developed as means for users and researchers to explore behavior and measure performance of file systems. These usually consist of putting file system under some kind of workload, gathering different measurements in the process and computing overall summary after the test is done. Measured parameters are usually bandwidth, latency and IO operations per second.

The benchmark should be highly configurable, so the researchers can simulate various workloads that would mimic real life scenarios. Stability and reproducibility of tests are also required, if the research is meant to be objective and to have some academic value about file system itself.

The standard workflow is tu run benchmark on a clean instance of operating system without other applications running, to avoid noise, and on clean instance of researched file system.

While this approach brings great results, with correct configuration of the benchmark, it only gives researchers a general idea about how does file system perform in early stage of it's usage.

However, there is a growing demand from users as well as from file system developers to explore behavior and performance of a file system, that has been used for longer period of time.

Prolonged usage of file system impel it to do more and more optimisations. When there is a lot of free space, there is generally no need for complicated approach. But with more files being deleted, created and expanded, the free space begins to be more and more fragmented. This then leads for files to be spread non contiguously across the physical device.

This introduces problem in regard of devices with moving parts, such as widely used hard disk drive. When working with I/O operations on this type of device, the fragmentation cause larger seek times, which is a time for magnetic heads to reach desired location. This is a great example of how file system has to be flexible to solve such problems as well as of problems that emerge after running file system for a longer period of time.

The ideal approach of research pointed on effect on age of file system would be to put file system under defined conditions for a few years or months, gathering information in the process. However, this is apparently impractical or even impossible, because the demand for results has to be satisfied in matter of weeks to respond to new versions of file systems, which are developed and released quite often.

One of the earliest ideas about how to deal with this problem would be to capture a snapshot of a file system that had already been used for a prolonged time. This approach of testing on such a snapshot, however, would probably lead to optimising for a very specific instance of old version of given file system.

If researchers want to predict behavior of a new version of file system after prolonged time of usage, they should be able to create a simulation, which would mimic the aging process, but in short period of time.

This can be technically implemented, because the idea is to run the simulation continuously under very heavy workload without using operations which does not alter file systems layout, therefore researchers should be able to compress the time of creating such an aged file system in a short period of time.

Unfortunately the demand from users and developers has yet not been met by standard studies. There has been few studies executed in last millenium, that shed some light on this topic, but overall, modern research is in hands of regular users, remaining in amateur sphere, lacking professional equipment to execute such research properly.

The aim of this thesis is to create standard, configurable workflow of creating and storing aged file system as well as a workflow to mesure its performance. Workflow for file system aging will use open source tool fs-drift to simulate aging process and then creates and stores file system image using system tools. Workflow for performance testing will then use created images to perform various tests.

These workflows will then be used to test XFS and EXT4 file systems on different storage media to show their strenghts and weaknesses in respect to long time runnign, while showing importance of this kind of testing.

The conclusion of research will show how does aging affect hard drive disks compared to solid state disks, what effect has aging on file system performance and what are the performance differences between XFS and EXT4 after long time usage.

The workflows created for this thesis will be included in Red Hat File system performance testing campaigns.

2 File systems and used tools

2.1 File systems

File system is a set of tools, methods, logic and structure to control how to store and retreive data on and from a storage, e.g. device. It is sometimes called a 'bookkeeper' of operational system. As an analogy to paper-based systems. Basic user-accesed units are called files, which could be clustered into directories.

The system stores files either continuously or scattered across device. The basic accessed data unit is called a block, which capacity can be set to various sizes. Blocks are labeled either as free or used.

Files which are non-continous are stored in form of extents, which is one or more blocks associated with the file, but stored elsewhere.

Information about how many blocks does a file occupy, as well as other information like date of creation, date of last access or access permissions is known as metadata, e.g. data about stored data. This information is stored separately from the content of files. On modern file systems, metadata are stored in objects called inodes (index nodes). Each file a file system manages is associated with an inode and every inode has its number in an inode table. On top of that the file system stores metadata unrelated to any specific file, such as information about bad sectors, free space or block availability.

(bit maps)

In this thesis, targeted file systems will be UNIX XFS and EXT4, which are main Red Hat supported file systems. These file systems belong to the group of journaling file systems.

Journaling file system keeps a structure called journal, which is a buffer of changes not yet commited to the file system. After system failure, these planned changes can be easily read from the journal, thus making the file system easily fully operational, and in correct and consistent state again.

2.2 XFS

XFS is a 64-bit journaling file system created by Silicon Graphics, Inc(SGI) in 1993. It is known for great performance in execution of paralel I/O operations, because of its architecture based on allocation groups.

Allocation groups are euqally sized linear regions within file system. Each allocation group manages its own inodes and free space, therefore increasing parallelism. Architecture of this design enables for significant scalability of bandwidth, threads, and size of file system, as well as files, simply because multiple processes and threads can access the file system simultaneously.

XFS allocates space as extents stored in pairs of B+ trees, each pair for each allocation group (improving performance especially when handling large files). One of the B+ trees is indexed by the length of the free extents, while the other is indexed by the starting block of the free extents. This dual indexing scheme allows for the highly efficient location of free extents for file system operations.

Prevention of file system fragmentation consist mainly of a feature called *delayed allocation* as well as online defragmentation(*xfs_fsr*), that can turururu

Delayed allocation, also called *allocate-on-flush* is a feature that, when a file is written to the buffer cache, substracts space from the free-space counter, but won't allocate the free-space bitmap. The data is held in memory until it have to be stored because of system call (such as *sync*). This approach improves the chance, that the file will be written in a contiguous group of blocks, avoiding fragmentation and reducing CPU usage as well.

2.3 EXT4

Ext4, also called fourth extended filesystem is a 48-bit journaling file system developed as successor of ext3 for Linux kernel, improving reliability and performance features.

Similary as xfs, ext4 use delayed allocation to increase performance, especially when in use with multiblock allocation and extent-based approach, also reducing fragmentation on the device. For cases of fragmentation that still occur, ext4 provide support for online defragmentation and *e4defrag* tool to defragment either single file, or whole file system.

2.4 FIO

Flexible Input/Output tool is a IO workload generator written by Jens Axboe. It is a tool well known for it's flexibility as well as large group of contributors and users.

2.5 Fs-drift

fs-drift is a workload aging test written by Ben England. It relies on randomly mixed requests generated by inner heuristic (according to parameters). These requests can be writes, reads, creates, appends or deletes.

At the beginning of run time, the top directory is empty, and therefore *create* requests success the most, other requests, such as *read* or *delete*, will fail because of lack of files and small probability of randomly choosing existing one.

Over time, as the file system grows, *create* requests began to fail and other requests succede more. Finally, file system will reach a state of equilibrium, when requests are equaly likely to execute. From this point, the file system would not grow anymore, and the test runs until one of the *STOP* conditions are met (specified with parameters).

Fs-drift is very flexible and can be used to simulate lots of different workloads by operating with various file sizes, request types and different kinds of random distribution.

2.6 File system images

To achieve consistency of results and to shorten testing time, file system images are used. Once the image is created, it can be stored for later use and replayed back on device. To save space, only metadata of created file system are used, since content of created files is random and therefore irrelevant. Replayed metadata point at various blocks on device, recreating fragmentation while seldom taking significantly less space. These images can be created by using tools developed to inspect file systems in case of emergency. For ext based file systems, there is e2image tool and for xfs, there is xfs_metadump. Both tools create images as sparse files, so compression is needed.

E2image tool can save whole ext based file system or just its metadata and offers compresion of image as well. Created images can be further compressed by tools such bzip2 or tar.
Creating compressed image using e2image:

```
e2image -Q $DEVICE $NAME.qcow2
```

Such images can be later replayed back on a device. From that point, file system can be mounted and revised.
Replaying compressed image:

```
e2image -r $NAME.qcow2 $DEVICE
```

Xfs_metadump saves XFS file system metadata to a file. Due to privacy reasons file names are obsfucated (can be disabled by -o parameter). As well as e2image tool, the image file is sparse, but xfs_metadump doesn't offer a way to compress the output. However, output can be redirected to stdout from where it can be passed to a compression tool. Creating compressed image using xfs_metadump:

```
xfs_metadump -o $DEVICE -|bzip2 > $NAME
```

Such images, when uncompressed can be replayed back on device by tool xfs_mdrestore. File system can be then mouned and inspected as needed:

```
xfs_mdrestore $NAME $DEVICE
```

3 Storage media

3.1 HDD and SSD

HDD is a rotational disk, which requires specific approach from kernel, to ensure the lowest possible seek time. Seek time is a time for moving parts of the device to find next relevant block of data. This affect overall performance greatly, because with large fragmentation, seek time becomes quite high.

As for SSD, this type of device does not have any moving parts, which make perform really well. One of the problems, however, is limited lifecycle of memory cells. SSD manufacturers deal with this problem by adding controler with its own scheduler, which make sure, no parts of the device are used significantly more than other parts.

When aging the filesystems, I expect for those grown on HDD to perform significantly slower after aging process, and I expect SSD filesystems not to be affected at all, or maybe significantly less.

3.2 SATA

3.3 SAS

3.4 HDD

3.5 SDD

4 Workflow

4.1 Workflow of image creating

Workflow of image creating is contained in the package drift_job. After extracting fs-drift, the main script starts python script, which handles the process of running fs-drift. Settings of fs-drift are passed as a parameter and are parsed inside the script. Before running the fs-drift, python daemon thread is created to log free space fragmentation periodically while fs-drift is running. After the aging process is done, overall fragmentation is computed.

After the aging process, the script use system tools to create and compress the image. Information about system is gathered as well and all the logs are archived and sent to data collecting server. Parameters available for drift_job:

1. -s|–sync, flag to signalise wheather or not to send data to server (usefull for developing purposes)
2. -m|–mountpoint
3. -d|–device
4. -r|–recipe, parameters to pass to fs-drift
5. -t|–tag, string to distinguish different tests

4.2 Fs-drift settings

As the creator states in README, to fill up a file system, maximum number of files and mean size of file should be defined such that the product is greater than the available space. So if the workload is supposed to fill 500GB of space, while having maximum file size of 1GB (therefore mean size is 500MB), maximum number of files should be much higher than 1000. Optimal approach is to define seemingly no upper limit to let the fs-drift fill the volume, therefore numbers as high as 10^8.

Parameter -t specifies the top directory, which will be used in test, in this workflow it is set to $MOUNTPOINT.

There is an option to specifiy user-defined file to use as a workload table, which is a desired percentual representation of operations in a workload. Since the goal of this workload is to create fragmented file system in a short time, read and rename operations are irrelevant. Therefore only create, append and delete have representation in this workload. The optimal results were reached when every operation had equal representation, e.g. 33%

The fs-drift allows directories up to defined level to create. The directory in which a file is directly affect its chance to be selected for a chosen operation, so by using only one directory, the equilibrium happens too fast, long before the file system is filled completely. Therefore we allow up to three levels of directories to be created.

One of the features of fs-drift that helps to simulate real-life file system access is enabling of moving random distribution. By using parameter -v, user can specify by how much will the number of files move for every operation.

Duration of the test is set to 5 hours so the test is usable for testing campaign without oversaturating of the servers.

On top of mentioned parameters, fs-drift can be set to send statistic logs such as response times of files and making statistics in defined period of time. These features are enabled in a workflow because of their importance in researching how the behaviour of file system changes over time.

4.3 Workflow of performance testing

Performance testing of created images is done by a package recipe_fio_aging. Upon instalation of necessary tools (libs, fio), the package finds and downloads coresponding file system image according to obtained parameters. As shown, images are stored compressed, therefore decompression is needed after download. Once these steps are succesfully completed, the image can be replayed on the device by using presented tools (e2image, xfs_mdrestore). If the image restoring completes succesfully, file system can be mounted and worked with exactly like it would be just after the aging process.

After image restoration, some files (according to given parameter) are deleted to create space for the FIO test to take place. The files to be removed are choosen randomly until desired amount of volume has been freed. By using this workflow, e.g. freeing some amount of space, we can simulate aged file system in various phases of aging by using just one image of a very fragmented file system.

When free space is reclaimed, FIO test will take place using parameters given to recipe_fio_aging. The overall space occupied by the test should not be larger than available space on the file system, otherwise the test will either fail completely or report incorrect results.

For statistical correctness, the FIO test can run several times in a row. After last iteration, the results are compressed and sent to data-collecting server.

Parameters available for recipe_fio_aging:

1. -s | –sync, flag to signalise wheather or not to send data to server (usefull for developing purposes)
2. -n | –numjobs, number of test repetitions. For statistical stability
3. -m | –mountpoint
4. -d | –device
5. -r | –recipe, parameters to pass to FIO test
6. -t | –tag, string to distinguish different tests

4.4 FIO settings

5 Testing environment

The aging process took place on these servers:

1. durden
2. Intel(R) Xeon(R) CPU, X2460 (2.80Ghz, ??? Cache,8 cores)
3. RAM 10GB
4. 4x300GB SAS HDD, 1x50GB SAS SSD

1. joker
2. Intel(R) Xeon(R) CPU, X2460 (2.80Ghz, ??? Cache,8 cores)
3. RAM 10GB
4. 4x300GB SAS HDD, 1x50GB SAS SSD

The system installed on machines is RHEL-7.2 with kernel 3.10.0-514.el7.x86_64

6 Results

The output of result generator is a htlm report summarising all information about system, links to raw data and charts of measured values.

6.1 Performance of aged file system

6.2 Differences betweem XFS and EXT4

6.3 Differences accross different storage

7 Conclusion

Here I will admit, that these results were not really surprising and ABSOLUTELY no breakthrough, however, as noone really research this branch of QE, the results are definitely a step further in this field.