Measure File Systems' Write Amplification

Zeyuan Hu
iamzeyuanhu@utexas.edu

1 Introduction

In this writeup, we empirically measure the write amplification for file system and detail the steps to reproduce the said result.

2 Background

We follow the concepts from "Designing Access Methods: The RUM Conjecture" [1] to define the space amplification and write amplification. There are two types of data: base data and auxiliary data. Base data means the main data stored in the system that we want to read or update. For example, the data to be written into the file system. Auxiliary data represents the data used for performance improvements on updating of base data. For example, when we append a block of data to a file, file system changes three blocks of data: the data block itself (base data), the inode block (auxiliary data), and the data bitmap (auxiliary data). Then the write amplification and space amplification can be defined as following:

$$\text{write amplification} = \frac{\text{update size to the auxiliary data} + \text{update size to the base data}}{\text{update size to the base data}}$$

$$\text{space amplification} = \frac{\text{space increase for auxiliary data} + \text{space increase for base data}}{\text{space increase for base data}}$$

For our appending a block of data to a file example, the write amplification is three because "update size to the auxiliary data" = 2 (the inode block, the data bitmap) and "update to the base data" is one (the data block). For the space amplification, since both updates to the inode block and the data bitmap are overwrite and "space increase for auxiliary data" =

0, thus both numerator and denominator is one. Thus, the space amplification is one.

3 Environment

We use a machine that has 4 Intel(R) Core(TM) i5 CPU @ 3.60GHz processors and 4GB of memory. The machine runs Ubuntu 14.04.5 LTS (kernel version 4.8.12). There are one HDD (e.g., `/dev/sda1`) and one SSD (e.g., `/dev/sdb1`) in the machine. We will use the SSD to run all of our experiments.

4 Measure the Write Amplification of file systems using Filebench

4.1 Setup the Filebench

Filebench [2] is a file system and storage benchmark that can generate various workloads. We use Filebench 1.5-alpha3 version to perform our experiments. We first install the Filebench to our environment:

```
$ wget https://github.com/filebench/filebench/releases/download/1.5-alpha3/
   filebench-1.5-alpha3.tar.gz
$ tar zxvf filebench-1.5-alpha3.tar.gz
$ cd filebench-1.5-alpha3/
$ ./configure --prefix=$HOME/filebench_install
$ make
$ make install
```

We run the varmail workload (e.g. `varmail.f`) shipped with the Filebench (e.g. `filebench_install /share/filebench/workloads`) to measure both the space and the write amplification. The varmail workload simulates a mail server, which is `fsync()` heavy [3]. We setup the varmail workload in the same way as mentioned in "Analyzing IO Amplification in Linux File Systems" [3] paper: 16 threads, total files 100K, mean file size 16 KB. We modify the `varmail.f` as following:

```
set $dir=/home/iamzeyuanhu/hzy/empty
set $nfiles=100k
set $meandirwidth=1000000
```

```
set $filesize=cvar(type=cvar-gamma,parameters=mean:16384;gamma:1.5)
set $nthreads=16
set $iosize=1m
set $meanappendsize=16k

define fileset name=bigfileset,path=$dir,size=$filesize,entries=$nfiles,
    dirwidth=$meandirwidth,prealloc=80

define process name=filereader,instances=1
{
  thread name=filereaderthread,memsize=10m,instances=$nthreads
  {
    flowop deletefile name=deletefile1,filesetname=bigfileset
    flowop createfile name=createfile2,filesetname=bigfileset,fd=1
    flowop appendfilerand name=appendfilerand2,iosize=$meanappendsize,fd=1
    flowop fsync name=fsyncfile2,fd=1
    flowop closefile name=closefile2,fd=1
    flowop openfile name=openfile3,filesetname=bigfileset,fd=1
    flowop readwholefile name=readfile3,fd=1,iosize=$iosize
    flowop appendfilerand name=appendfilerand3,iosize=$meanappendsize,fd=1
    flowop fsync name=fsyncfile3,fd=1
    flowop closefile name=closefile3,fd=1
    flowop openfile name=openfile4,filesetname=bigfileset,fd=1
    flowop readwholefile name=readfile4,fd=1,iosize=$iosize
    flowop closefile name=closefile4,fd=1
  }
}

echo "Varmail Version 3.0 personality successfully loaded"

run 60
```

File System	Write Amp.	
Journal On?	Yes	No
ext2	5.1	N/A
ext4	4.7	3.6
xfs	3.3	N/A
f2fs	2.4	N/A
btrfs	5.7	N/A

Table 1: Write amplification for different file systems on SSD . We measure whether journaling will cause a significant incrase in the write amplification of the file system.

4.2 Way of measurement and the result

We measure `ext2`, `ext4`, `xfs`, `f2fs`, `btrfs` mounted on SSD (e.g. `/dev/sdb1`). To calculate the write amplification empirically, we use `iostat` [4], `du` [5], and `strace` [6] along with the customized Filebench.

We modify the Filebench [1] to expose two additional statistics: the total write of the workload and the `iostat` number difference after the preallocation of the files. As shown in the varmail workload setup, filebench will initially create a fileset and allocate 80% of the files before the actual workload run. Thus, during the calculation of write amplification, we want to exclude the writes done by the preallocation. We expose the total write of the workload from Filbench because it will give an accurate measure of the number of writes issued from the workload (i.e., the denominator of the write amplification calculation).

For each file system, we run the command `iostat -d /dev/sdb1` first to record `kB_wrtn`, which represents the total number of kilobytes written on the device. Then, we run the filebench and monitor the whole process using the `strace`. We filter the `write` system call. After the finish of the process, we run the `iostat -d /dev/sdb1` again to record `kB_wrtn` number, calculate the number of bytes written through the `write` system call. Then, we calculate the space and write amplification as following:

$$\text{write amplification} = \frac{\texttt{kB\_wrtn} \text{ number difference} - \text{write done by the preallocation}}{\text{total writes done by the workload}}$$

We automate the whole process through a script [2]. The result is shown in Table 1.

[1] https://github.com/xxks-kkk/filebench/tree/amp

[2] https://github.com/xxks-kkk/Code-for-blog/blob/master/2018/emmett/filebench/benchmarks/filebench/strace.sh

File System	`iostat` difference (KB)	
Journal On?	Yes	No
ext4	3277472	3272320

Table 2: The number of KB written to the disk measured by `iostat` on ext4

5 Measure the Write Amplification of file systems using Git-benchmark

Betrfs [7] uses a Git benchmark, which is described in the "File Systems Fated for Senescence? Nonsense, Says Science!" [8]. Essentially, the benchmark clones the Linux repository and perform `git pull` to patch the code for a fixed amount of times. We're interested to see the amount of data written to the disk shown by the `iostat`.

We modified the Git benchmark to include `sync` after each pull and use `fstrim` to trim SSD during the inital file system setup. We perform 20 pulls for the experiment [3]. The result is shown in Table 2.

References

[1] M. Athanassoulis, M. S. Kester, L. M. Maas, R. Stoica, S. Idreos, A. Ailamaki, and M. Callaghan, "Designing access methods: The rum conjecture.," in *EDBT*, vol. 2016, pp. 461–466, 2016.

[2] V. Tarasov, E. Zadok, and S. Shepler, "Filebench: A flexible framework for file system benchmarking," *USENIX; login*, vol. 41, 2016.

[3] J. Mohan, R. Kadekodi, and V. Chidambaram, "Analyzing io amplification in linux file systems," *arXiv preprint arXiv:1707.08514*, 2017.

[4] "iostat(1) - linux man page." `https://linux.die.net/man/1/iostat`.

[5] "du(1) - linux man page." `https://linux.die.net/man/1/du`.

[6] "strace(1) - linux man page." `https://linux.die.net/man/1/strace`.

[3] Code: https://github.com/xxks-kkk/Code-for-blog/tree/master/2018/emmett/filebench/benchmarks/git-benchmark

[7] W. Jannen, J. Yuan, Y. Zhan, A. Akshintala, J. Esmet, Y. Jiao, A. Mittal, P. Pandey, P. Reddy, L. Walsh, *et al.*, "Betrfs: A right-optimized write-optimized file system.," in *FAST*, pp. 301–315, 2015.

[8] A. Conway, A. Bakshi, Y. Jiao, W. Jannen, Y. Zhan, J. Yuan, M. A. Bender, R. Johnson, B. C. Kuszmaul, D. E. Porter, *et al.*, "File systems fated for senescence? nonsense, says science!," in *FAST*, pp. 45–58, 2017.