File Systems Performance Analysis

Benchmarking project

for the lecture
Computer Performance Analysing And Benchmarking

lectured by Prof. Thomas M. Stricker

at Swiss Federal Institute of Technology

Written by Stefan Rondinelli

CONTENTS

0 Introduction

Every operating system has its own file systems. For example Windows uses FAT16(File Allocation Table 16(bits)), FAT32(File Allocation Table 32) and NTFS(Windows NT File System), Linux uses Minix fs, Extended fs and ext2 and a Mac has its (discarded) MFS(Macintosh file system) and HFS(Hierarchical File System) file systems. Sometimes file systems of other operating systems are supported what for example is desired in a dual boot system (e.g. Linux and Windows).
When using such a dual boot machine with Linux and Windows, the performance of a file system could be one of the considerations when choosing what file systems to use for the different partitions.
In this project I am going to benchmark some file systems and finally to analyze the gathered data statistically as learned in the lecture.

1 The Environment

1.1 The Hardware

I used the following hardware for the performance analysis:

- CPU: AMD K6II 450Mhz (has a 64kB cache)
- RAM: 64MB SDRAM
- Hard disk 1: Western Digital Caviar 36400 (6 GB)
- Hard disk 2: Western Digital Caviar 33200 (3 GB)
- Disk controller: IDE for both hard disks

To have the same conditions these components affecting the I/O speed must be the same ones for all the measurements of the performance of the different file systems otherwise the differences in the measured data would rather be due to unequal hardware then to the different implementation of a file system.

The first hard disk where I installed the operating system was attached to the first IDE controller and the second hard disk which was used exclusively for the performance analysis was attached to the second IDE controller. No other peripheral devices (i.e. other disks or CDROM drives) where attached to the controllers.

The computer I originally used had more RAM. I only left the smallest RAM module inside because to in order get reasonable results the file size of the workload should be at least four times the size of the available memory and I didn't want to read and write gigabytes of data on the disk for each single test. This is because the operating system caches large parts of the file and the benchmark will end up doing very little I/O.

1.2 The Software

I installed a Linux as operating system because that's the only operating system which supports almost every file system. The operating system was installed on the first hard disk having three partitions: A boot partion with the kernel (starting at the first cylinder) of 16MB followed by the swap partition of 192 MB and finally a third partition filling the rest of the hard disk for the programs. I didn't use any graphical interface (X Server) to have as little perturbation as possible for the benchmark.

On the second hard disk I made a partition of 1 GB formatted with the file system to be tested for each benchmark.

I chose to test the following file systems: **FAT16, FAT32** and **ext2**. FAT16 and FAT32 might be the most used file systems because Windows is the operating system that is installed on most PCs. Ext2 is the most used file system for Linux. Originally I also wanted to test some other file systems but this was not possible for the following reasons:
HFS (the file system used by a Macintosh) would as well have been interesting as Macintoshes are used by many people but unfortunately the distribution of Linux I used didn't have a formatting program for this file system (although it's been supported since Kernel 2.2.x)
Minix fs (the file system of a Minix operating system) allows at most 64K blocks. This was not enough to have a sufficient big partition for the benchmark.

As benchmarking program I used Bonnie (available at www.textuality.com). There was a makefile to compile it which didn't work. So I compiled it using the (gnu) gcc compiler. Bonnie does the following:

Description of Bonnie from the internet [bonnie]:

1. Sequential Output

1.1 Per-Character

The file is written using the *putc()* stdio macro. The loop that does the writing should be small enough to fit into any reasonable I-cache. The CPU overhead here is that required to do the stdio code plus the OS file space allocation.

1.2 Block

The file is created using *write(2)*. The CPU overhead should be just the OS file space allocation.

1.3 Rewrite

Each `Chunk` (currently, the size is 16384) of the file is read with *read(2)*, dirtied, and rewritten with *write(2)*, requiring an *lseek(2)*. Since no space allocation is done, and the I/O is well-localized, this should test the effectiveness of the file system cache and the speed of data transfer.

2. Sequential Input

2.1 Per-Character

The file is read using the *getc()* stdio macro. Once again, the inner loop is small. This should exercise only stdio and sequential input.

2.2 Block

The file is read using *read(2)*. This should be a very pure test of sequential input performance.

3. Random Seeks

This test runs `SeekProcCount` (currently 4) processes in parallel, doing a total of 4000 *lseek()*s to locations in the file computed using by *random()* in bsd systems, *drand48()* on sysV systems. In each case, the block is read with *read(2)*. In 10% of cases, it is dirtied and written back with *write(2)*.

The idea behind the `SeekProcCount` processes is to make sure there's always a seek queued up.

AXIOM: For any Unix file system, the effective number of *lseek(2)* calls per second declines asymptotically to near 30, once the effect of caching is defeated. [I wrote the previous sentence in about 1988, and it's a bit better now, but not much]

The size of the file has a strong nonlinear effect on the results of this test. Many Unix systems that have the memory available will make aggressive efforts to cache the whole thing, and report random I/O rates in the thousands per second, which is ridiculous. As an extreme example, an IBM RISC 6000 with 64 Mb of memory reported 3,722 per second on a 50 Mb file. Some have argued that bypassing the cache is artificial since the cache is just doing what it's designed to. True, but in any application that requires rapid random access to file(s) significantly larger than main memory which is running on a system which is doing significant other work, the caches will inevitably max out. There is a hard limit hiding behind the cache which has been observed by the author to be of significant import in many situations - what we are trying to do here is measure that number.

End of [bonnie]

2 Performing the Benchmark

2.1 The Proceeding

After the partition was made (the partition always started at the first cylinder as the speed of reads and writes is different at different locations on the disk) and formatted with the file system to be tested, it was mounted at /test in the directory tree.

For ext2 I used Linux to partition and format whereas for FAT16 and FAT32 I used MS-DOS. I just controlled with Linux' fdisk if the FAT partitions started at cylinder 1 as MS-DOS' fdisk didn't provide this information.

The following command started the benchmark:

./Bonnie -d /test -s 320 >> out.txt

The option d specifies the directory to read and write the scratch file, whereas with the option s the size of the scratch file (in MB) can be determined (otherwise Bonnie would use 100 MB). 320 MB is five times the amount of available memory that therefore should be enough (as explained before). The output was appended to the out.txt file.

I made each test twice, once with 256 MB (4 times the size of the available memory) and once with 320 MB (5 times), to see if the results where more or less the same. So I could supervise the measurements and see if the workload was big enough.

2.2 The Output

Here is the raw (I formatted it a bit afterwards (added some lines) to make it more readable) output of the benchmark:

```
ext2
     -------Sequential Output-------- ---Sequential Input-- --Random--
     -Per Char- --Block--- -Rewrite-- -Per Char- --Block--- --Seeks---
 MB K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU  /sec %CPU
256  2843 98.3  8162 89.3  3352 58.2  2615 98.1  7584 96.4  75.5 12.3
     -------Sequential Output-------- ---Sequential Input-- --Random--
     -Per Char- --Block--- -Rewrite-- -Per Char- --Block--- --Seeks---
 MB K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU  /sec %CPU
320  2876 98.8  8002 88.8  3321 57.1  2629 97.6  7601 96.2  72.9 11.4
------------------------------------------------------------------
------------------------------------------------------------------
FAT16
     -------Sequential Output-------- ---Sequential Input-- --Random--
     -Per Char- --Block--- -Rewrite-- -Per Char- --Block--- --Seeks---
 MB K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU  /sec %CPU
256  2616 90.6  6199 77.1  2810 75.2  2527 97.6  7150 99.2  77.8 13.6
     -------Sequential Output-------- ---Sequential Input-- --Random--
     -Per Char- --Block--- -Rewrite-- -Per Char- --Block--- --Seeks---
 MB K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU  /sec %CPU
320  2607 90.6  6208 78.1  2812 73.6  2529 97.5  7172 98.0  72.2 15.5
------------------------------------------------------------------
------------------------------------------------------------------
FAT32
     -------Sequential Output-------- ---Sequential Input-- --Random--
     -Per Char- --Block--- -Rewrite-- -Per Char- --Block--- --Seeks---
 MB K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU  /sec %CPU
256  2317 75.4  3938 40.7  1890 76.2  2399 88.7  5659 71.3  70.8 76.1
     -------Sequential Output-------- ---Sequential Input-- --Random--
     -Per Char- --Block--- -Rewrite-- -Per Char- --Block--- --Seeks---
 MB K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU  /sec %CPU
320  2325 73.9  3781 36.9  1779 79.6  2407 88.6  5030 62.4  60.0 81.5
------------------------------------------------------------------
```

3 Statistical Analysis of the Data

3.1 Computation of Effects

Computation of effects						
Workload	**ext2**	**FAT16**	**FAT32**	**Row sum**	**Row mean**	**Row effect**
Output per char	2876	2607	2325	7808	**2602.67**	-1469.07
Output per block	8002	6208	3781	17991	**5997.00**	1925.27
Rewrite	3321	2812	1779	7912	**2637.33**	-1434.40
Input per char	2629	2529	2404	7562	**2520.67**	-1551.07
Input per block	7601	7172	5030	19803	**6601.00**	2529.27
Column sum	24429	21328	15319	61076		
Column mean	**4885.80**	**4265.60**	**3063.80**		**4071.73**	
Column effect	814.07	193.87	-1007.93			

Anova table						
Component	**Sum of squares**	**Percentage of variation**	**Degrees of freedom**	**Mean squares**	**F-Computed**	**F-Table 90 / 95 / 99 %**
y	313042288					
y..	248685185					
y-y..	64357103	100.00	14			
Filesystem	8581092	13.33	2	4290546	6.1	3.1 / 4.4 / 8.7
Action	50175929	77.96	4	12543982	17.9	2.8 / 3.8 / 7.0
Errors	5600082	8.70	8	700010		

I made a two-factor full factorial design without replications. One factor is the file system having 3 levels. The other factor is the action having 5 levels namely actions such as reading and writing that where tested by the benchmark.

Before looking at the result, let's examine the Anova (=Analysis of variance) table to see if the factors are assumed to explain a significant fraction of the variation. If the computed ratio mean of squares of a factor divided by the mean of squares of errors (column "F-Computed") is greater than value of the found in the table of quantiles of F-variates (column "F-Table") than the fraction of the variation will be significant.
For the file system this is the case at a confidence level between 95% and 99% as 6.1(F-Computed) lies between 4.4 and 8.7(F-Table). For the factor "Action" it's over 99% as 17.0 is bigger than 7.0 (This is not surprising because of the big overhead writing single characters instead of entire blocks).

Now let's see the results. The average I/O speed of all file systems and actions is 4071 kB/s. Ext2 is on average 814 kB/s faster, FAT16 is 194 kB/s faster and FAT32 is 1008 kB/s more slowly on average.

3.2 Confidence intervals

Standard deviation of errors:

$$S_e = \sqrt{MSE} = \sqrt{700010} = 836.67$$

Standard deviation of the grand mean **m**:

$$S_m = S_e / \sqrt{ab} = 836.67 / \sqrt{15} = 216.03$$

Standard deviation of $\boldsymbol{a}_j$'s :

$$S_{a_j} = S_e\sqrt{(a-1)/ab} = 836.67\sqrt{\frac{2}{15}} = 305.51$$

Standard deviation of $\boldsymbol{b}_j$'s :

$$S_{b_j} = S_e\sqrt{(b-1)/ab} = 836.67\sqrt{\frac{4}{15}} = 432.05$$

The degrees of freedom for the errors are (a-1)(b-1) = 2*4 = 8 and therefore

- for a 90% confidence interval $t_{[0.95;8]} = 1.86$
- for a 95% confidence interval $t_{[0.975;8]} = 2.306$

Confidence interval for the effects				
Parameter	Mean effect	Standard deviation	Confidence interval 90%	Confidence interval 95%
m	4071,73	216,03	(3670,4474)	(3574,4570)
File systems				
ext2	814,07	305,51	(246,1382)	(110,1519)
FAT16	193,87	305,51	(-374,762)	(-511,898)
FAT32	-1007,93	305,51	(-1576,-440)	(-1712,-303)
Actions				
Output per char	-1469,07	432,05	(-2273,-665)	(-2465,-473)
Output per block	1925,27	432,05	(1122,2729)	(929,2922)
Rewrite	-1434,40	432,05	(-2238,-631)	(-2431,-438)
Input per char	-1551,07	432,05	(-2355,-747)	(-2547,-555)
Input per block	2529,27	432,05	(1726,3333)	(1533,3526)

<div align="center">95% Confidence interval</div>
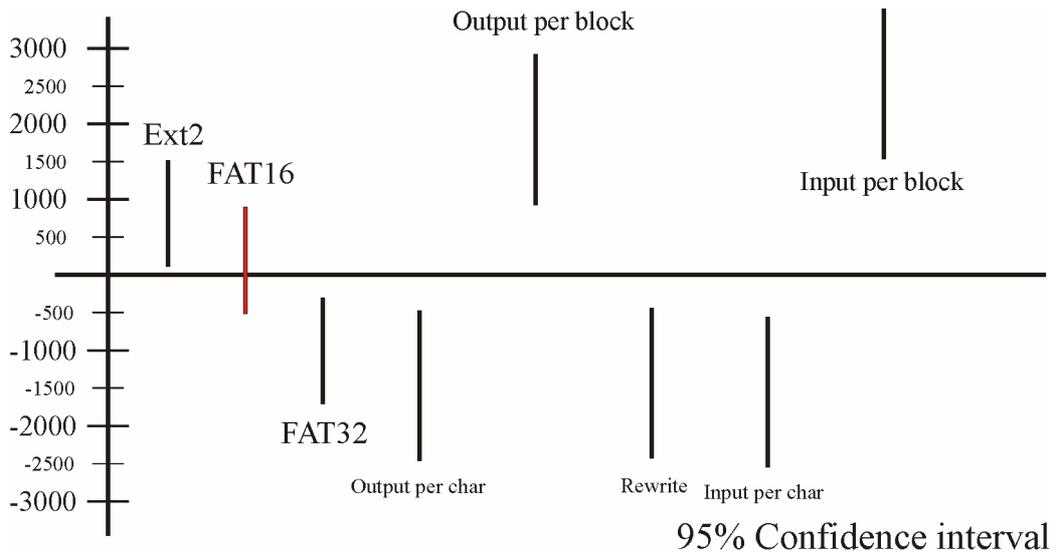
We can see in the graph that at a confidence level of 95% ext2 is faster than the average I/O speed and FAT32 is more slowly. However we cannot say at this confidence level that FAT16 is significantly faster or more slowly nor can we say something about that at a confidence level of 90% what can be seen in the table (The range (-374,762) includes zero).

3.3 Are the File Systems Significantly Different?

Using the formulas of contrast we can now calculate the confidence intervals for the differences of the three file systems.

For two file systems x and y:

Mean value = $\displaystyle\sum_{j=1}^{a} h_j \overline{y.}_j \Rightarrow \boldsymbol{a}_x - \boldsymbol{a}_y$

Standard deviation = $s_e \sqrt{\displaystyle\sum_{j=1}^{a} \frac{h_j^2}{b}} \Rightarrow s_e \sqrt{\dfrac{2}{5}}$

The following α's correspond the following file systems:

$\boldsymbol{a}_1 = ext2$

$\boldsymbol{a}_2 = FAT16$

$\boldsymbol{a}_3 = FAT32$

Confidence interval for the differences between the file systems					
		$t_{[1-a/2;8]}=$	0.889	1.108	1.397
Parameter	Mean value	Standard deviation	Confidence interval 60%	Confidence interval 70%	Confidence interval 80%
$a_1 - a_2$	620,20	1322,89	(-556,1796)	(-846,2086)	(-1228,2468)
$a_1 - a_3$	1822,00	1322,89	(646,2998)	(356,3288)	(-26,3670)
$a_2 - a_3$	1201,80	1322,89	(26,2378)	(-264,2668)	(-646,3050)
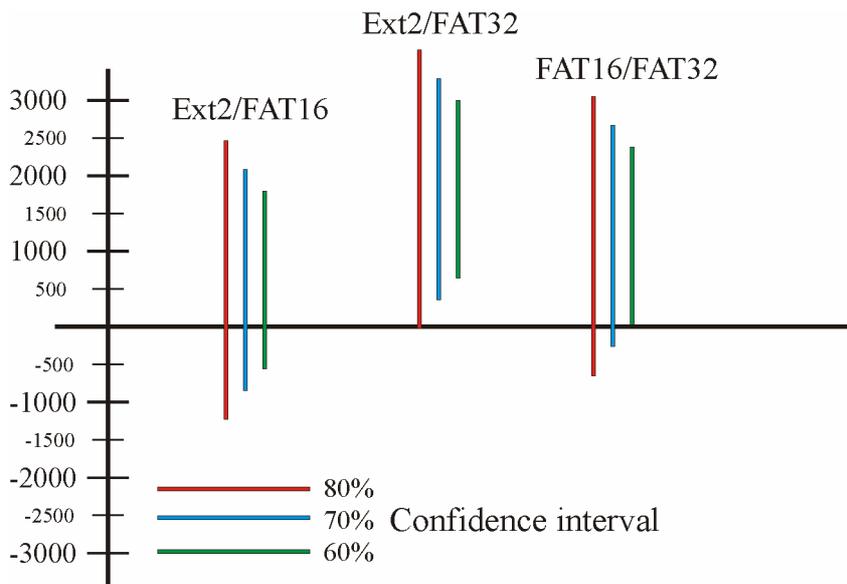
FAT16 and ext2 cannot really be estimated different in performance. Not even at a low confidence level of 60% they are significantly different.

FAT16 and FAT32 have almost the same performance. Being significantly different at a confidence level of 60% is rather a tendency.

At last ext2 and FAT32 are significantly different at a confidence level of almost 80%. I would say this should be enough to be able to say that ext2 is faster than FAT32.

References

[bonnie] http://www.textuality.com/bonnie/intro.html

The "recipes" for the statistical analysis of the following book where used:

"The Art of Computer Systems Performance Analysis" by Raj Jain
(John Wiley & Sons, INC.)