Optimization of Minix File System by not writing zero filled blocks to disk

Marcin Byra
University of Warsaw,
Faculty of Mathematics, Comp. Sc. and Mechanics
Banacha 2
Warsaw, Poland
mb347056@okwf.fuw.edu.pl

ABSTRACT

The optimization of Minix File System (MFS)[1] was done to increase write speed and minimize disk usage by avoiding to write blocks which are filled with bytes of value zero to disk. This paper describes how this little adjustment noticeably accelerates disk operations when dealing with files with significant percentage of zero bytes (such as virtual machines) and some boost with various standard files. We show comparison of write speed of machines with standard and modified MFS and discuss advantages of introducing the proposed optimization.

1. MOTIVATION

When dealing with virtual machines, significant areas of allocated disk space are empty (filled with bytes of value zero). This areas are intended as user space. However, it is not certain if they are ever fully used, and even if, there is no need to spare real disk space right from the beginning. Also files of various types can contain some empty areas which involves potentially wasted space and time overhead every time when performing copy or move.

Implementation of MFS which is similar on EXT3 file system provides that space allocated for new file does not have to be continuous. This let us to introduce optimization based on the idea of detecting zero-blocks and ignoring them in terms of writing. This does not interfere with file size seen by user for the system marks these blocks as null pointed.

2. IMPLEMENTATION

The change in a source code was little. The only modified function was rw_chunk in read.c in MFS source directory. At the beginning we checked if a block to be written was full of zeros by copying bytes to write to a local buffer and scanning it byte by byte. Then, the standard implementation of rw_chunk allocates a new block or overwrites existing block with new data. In both cases we just return from function if input block consists only of zeros. Further possible optimization to free existing block when it is to be filled by zeros instead of just returning from function is possible. Regardless it can improve used space, it would also bring time overhead. Because of that we skip this part

SO '17 Warsaw, Poland

ACM ISBN 123-4567-24-567/08/06…$99.00
DOI: 10.475/123_4

Table 1: Writing tests

	Original [s]	Modified [s]	Original/Modified
Test 1	0,78	0,49	159%
Test 2	1,49	0,87	171%
Test 3	7,47	1,89	394%
Test 4	11,16	2,94	379%
Test 5	36,77	39,31	94%
Test 6	4,27	4,90	87%

3. COMPARISON TESTS

We conducted a number of tests on both original machine and with modified file system. The tests were as follows: (by empty file we mean file of given size filled with bytes of value zero)

1. Creating a big empty file (300MB)
2. Copying a big empty file (300MB)
3. Creating 10 small empty files (10MB)
4. Copying 10 small empty files (10MB)
5. Copying a directory with many files of different type - text, pdf, photos etc. (200MB)
6. Copying a big file (250MB)

Table 1 compares time of these operations on original system and after the modification. All tests were repeated 5 times; visible values are averaged.

4. DISCUSSION

There was a need for testing and comparing modified system in two ways: to examine time difference when performing write operations on various files and to check if the system is aware of intended size of files regardless of saving real space.

4.1 Performance

The tests can be divided into two categories: first part examining file system's behavior when dealing with files mostly filled with zeros; last two tests measure time of operations performed on different standard files (hundreds of text source files, documents, multimedia).

The purpose of first four tests was to confirm that implementation works as expected and to examine how large the acceleration is. The difference of file manipulation is noticeable even without time measure. For many smaller files speed-up varies from about 160% and 170% when writing to and copying respectively. Results of operations on one heavier file are even greater: on the modified MFS time differences reached almost 400%. It seems natural that the more files are to be written the lesser speedup is noticed.

The second part of tests was performed to measure changes in system performance when dealing with standard files which are neither zero-filled files nor a kind with much empty spaces such as virtual machines etc. Analysis of the results leads to an observation that implemented opitimizations slow down write and copy on both many smaller and one bigger file. The effect is less visible than speed-up for empty files though: 94% and 87% of clear system for copying small files and big one respectively. The reason of this behaviour seems to be obvious: in our implementation rw_chunk must examine all bytes to write to check if there are non-zero values. Despite of the processing taking place in memory the overhead of examining each block to write is significant.

4.2 Space

The system displays correct sizes of files despite of their real physical size on disk. The space on disk is saved because new blocks with bytes of value zero are not allocated. The system was checked and works properly after the modification of MFS.

5. CONCLUSION

We have presented a possible optimization of MFS based on detecting zero-block during write operations and not writing them to disk. Simple this implementation may seem, the tests showed significant differences in terms of writing and copying files. The results suggest that when dealing with files with many empty blocks the improvement of system performance is very large, reaching almost 400% for big files; more space on disk is another important advantage. Nevertheless, this patch cannot be referred as universal. There is a minor performance decrease concerning standard files. In normal cases most of file operations are performed on such non-zero-filled files, hence these small slow-downs can affect whole system to work less efficient.

6. REFERENCES

[1] A. S. Tannenbaum. *Operating Systems: Design and Implementation.* Pearson Education, 3nd edition, 1987.