VU Research Portal

Immediate Files

Mullender, S.J.; Tanenbaum, A.S.

document version
Publisher's PDF, also known as Version of record

Link to publication in VU Research Portal

Immediate Files

SAPE J. MULLENDER AND ANDREW S. TANENBAUM

Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, The Netherlands

SUMMARY

An efficient disk organization is proposed. The basic idea is to store the first part of the file in the index (inode) block, instead of just storing pointers there. Empirical data are presented to show that this method offers better performance under certain circumstances than traditional methods.

In many transaction-oriented computer systems, the performance bottleneck is in accessing disk files. Consequently, when a file system is being designed, careful thought should be given to trying to minimize the number of disk accesses. In this paper we discuss some measurements we have made on an actual file system, then we look at a new kind of file organization suggested by these measurements, and finally we compare the performance of the new file organization to that of the UNIX* operating system.

Table I. Percentage of files smaller or equal to the indicated length

File length	Percentage	File length	Percentage
1	1·79	1024	48·05
2	1·88	2048	60·87
4	2·01	4096	73·51
8	2·31	8192	84·97
16	3·32	16,384	92·53
32	5·13	32,768	97·21
64	8·71	65,536	99·18
128	14·73	131,072	99·84
256	23·09	262,144	99·96
512	34·44	524,288	100·00

While designing a free-standing transaction- (as opposed to connection-) oriented file server for the Amoeba[1] distributed operating system, we made some measurements of file sizes on our departmental UNIX system. The results are summarized in Table I. For example, 60·87 per cent of the 19,978 files on the disk are 2048 bytes or less. The conclusion to be drawn from this data is simple: most files are short.

As an example of how file systems are typically organized, consider the UNIX file system.[2] Associated with each file is a 64-byte data structure called an *inode*. The

---

*UNIX is a Trademark of Bell Laboratories.

inode contains some bookkeeping and accounting information plus 13 disk addresses. These disk addresses occupy three bytes each. Each of the first 10 of these can point to a disk block containing some data. If disk blocks are n bytes long, files up to length $10n$ bytes can be accommodated in this way. For longer files, the eleventh address points to a disk block, called an *indirect block* containing $n/4$ disk block addresses. (A disk block address occupies four bytes.) For files larger than $(10 + n/4)n$ bytes, the twelfth disk address in the inode points to a *double indirect* disk block that points to $n/4$ additional indirect blocks. Finally, for huge files, the thirteenth disk address in the inode points to a *triple indirect* block.

The inodes are located contiguously in a sequence of blocks near the start of the disk. When a file is referred to by its ASCII name, the directory system maps the string onto an inode number, which is then used as an index into the inode block to find the inode. Thus, for files up to length $10n$ bytes, two disk access are required, one to fetch the inode and one to fetch the data block. (In a connection-oriented file system, the inode need only be fetched once, when the file is opened, but in a transaction-oriented file server that erases its tables after each request has been replied to, the inode must be refetched on each transaction.)

The combination of short files and the need to make two disk accesses suggests another possible file organization: expand the inode to a full disk block, and put the first part of the file in it. We call this an *immediate block*, in analogy with an immediate operand to a machine instruction. If the block size is 2048 bytes, some 60 per cent of the files can be accessed in only one disk operation. For larger files, access to parts of the file outside the immediate part would require the same number of disk accesses (two, three, or four) as in UNIX.

Before we describe our file organization in more detail let us compare the number of disk accesses required to read every byte in our sample of 19,978 files for UNIX and for our proposed file organization. We have also computed the storage efficiency using both immediate files and UNIX, again for the measured file length distribution. It is important to use actual length distributions because the whole concept of an immediate file only makes sense in the light of empirical data showing that short files are common. The results of these calculations are given in Table II. The two columns

Table II. Disk accesses and storage efficiency for various block sizes

Block size	Disk accesses per read/write		Percentage disk storage wasted	
	UNIX	Immediate files	UNIX	Immediate files
512	2·12	1·69	8·3	13·9
1024	2·06	1·46	13·3	14·4
2048	2·02	1·29	22·2	22·0
4096	2·01	1·16	36·7	36·6

labelled 'Percentage disk storage wasted' were computed by $(A - L)/A$, where A is the amount of space allocated to files and inodes and L is the total length of the files (i.e. the user data).

The conclusion to be drawn from this study is that immediate files can provide improved response times for transaction-oriented file servers. If the block size is small, the response time is improved at the cost of less efficient use of storage, but

when the block size becomes 2048 bytes or more, immediate files are a little less wasteful than UNIX files. Furthermore, we conclude that the relative advantage of immediate files over the UNIX organization increases with increasing block size.

We shall now describe the organization of immediate files in more detail. When a file is created, an inode block is allocated. Unlike UNIX, inodes need not reside at the beginning of the disk, but may be located anywhere. The last 48 bytes of an inode block are reserved for the inode. The rest of the block is used for immediate data. The structure of the inode can be exactly as in UNIX, with the exception that only 24 bytes are available for block pointers, whereas UNIX has 40 bytes worth of pointer space. These 24 bytes are used for 5 pointers to direct blocks, and one pointer each to an indirect block, a double indirect block, and a triple indirect block, giving 8 pointers altogether. Each pointer block can contain $(n{-}48)/4$ pointers, and a data block can contain $n{-}48$ data bytes, since the last 48 bytes of each block (the inode space) remain unused for pointers or data. This space may be used to hold recovery information for possible disk crashes and *hints* to make sequential file access even faster.[3] The file organization is illustrated in Figure 1.
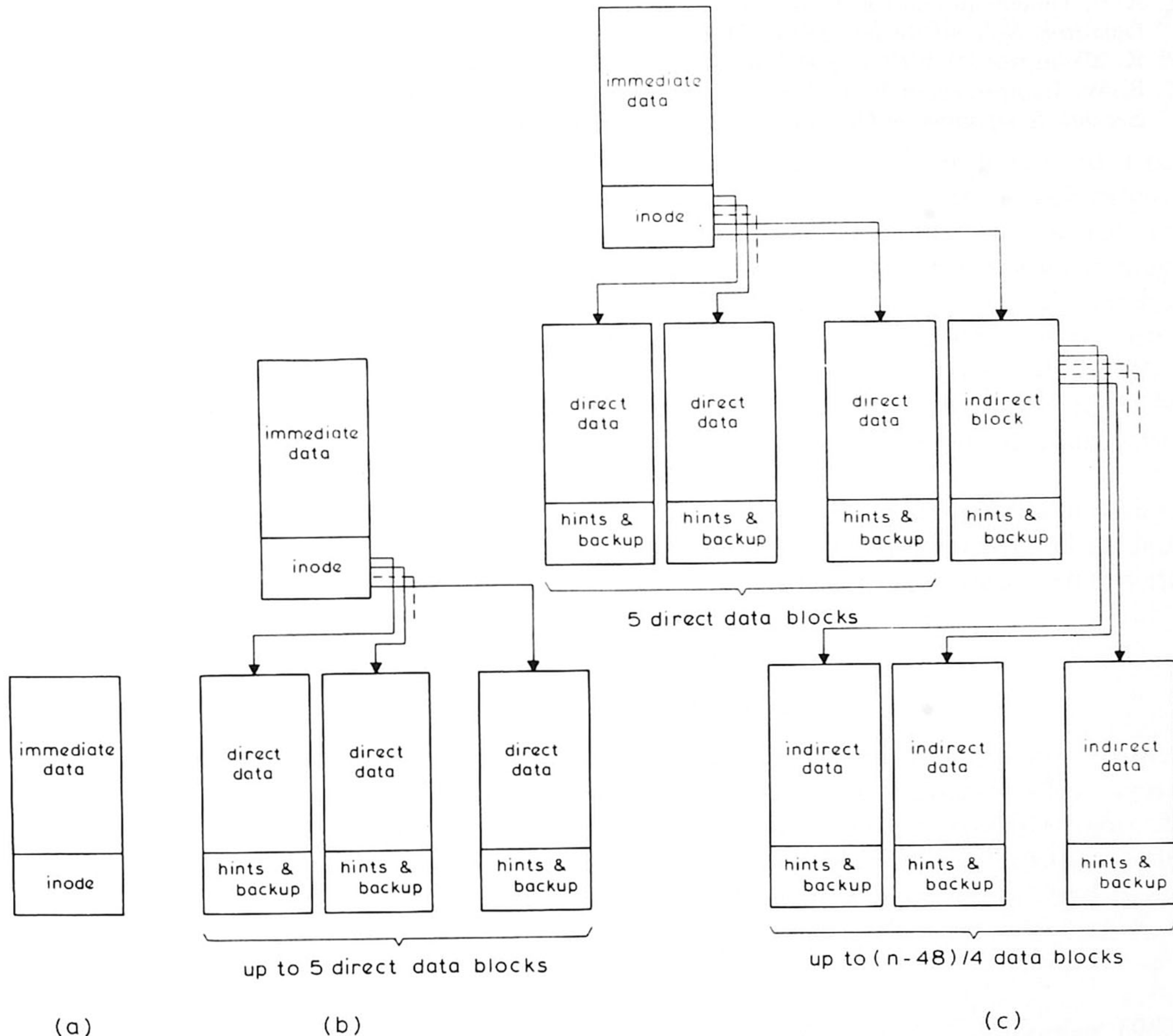

Figure 1. *(a) immediate file, (b) direct file, (c) indirect file*

If we assume a block size of 2048, then 2000 bytes are available in every block for pointers or data. Of every file, the first 2000 bytes are in the inode block, the next 10,000 bytes are in five direct blocks, pointed to by the five direct pointers in the inode. The next 1,000,000 bytes are in 500 indirect blocks, pointed to by 500 pointers in a pointer block, pointed to by the indirect pointer in the inode. There can be $500 \times 500 \times 2000$ or half a billion double indirect bytes, and $500 \times 500 \times 500 \times 2000$ or 250 billion triple indirect bytes. Since most disk drives are less than 500 Megabytes, it is also possible, when the blocksize is 2048 bytes or more, to use inodes with six direct blocks, one indirect block, one double indirect block, and *no* triple indirect block, since it is not needed then.

REFERENCES

1. A. S. Tanenbaum and S. J. Mullender, 'An overview of the Amoeba distributed operating system', *Operating Systems Review*, **15** (3), 51–64 (1981).
2. K. Thompson, 'UNIX implementation', *Bell System Tech. J.*, **57**, 1931–1946 (1978).
3. B. W. Lampson and R. F. Sproull, 'An open operating system for a single user machine', *Proc. Seventh Symposium on Operating System Principles*, 98–105 (1979).