A "LOG-STRUCTURED" FILE SYSTEM

Who needs a disc ? As memory gets cheaper, and processors get faster, and the hardware gets more and more reliable, why don't we just keep all the file material we need permanently in memory ? It's really a matter of insurance : if the system should break down, data stored on the disc will be preserved. Even this reason may well become unconvincing if software reliability also continues to increase, and the sole use of the disc will then be as an archive for comparatively infrequently used information.

In that case, it would be sensible to reassess the way in which we store information on the disc. Here's an interesting argument which follows this line through and has achieved respectability over the last few years. Read the original publication[IMP26] if you want more detail – it's admirably comprehensible. Just to show that it isn't a joke, further publications[SUP6, IMP27] record later developments; the system has actually been implemented, and works.

THE PROBLEM.

As the disparity in speed between processors and discs increases, the first difficulty is in the bandwidth of the disc path. One way round this for some purposes is simply to use a **disc array** – in effect, to use a lot of disc drives in parallel. Each file can be distributed between the drives in some efficient way (called *striping*) so that the total bandwidth of the combined channels is sufficient. (We describe disc arrays in more detail in the next chapter.)

But that simply moves the problem one step along : now we want to know how most effectively to use such techniques to provide the disc service we want. To answer the question, we have to know something about the patterns of use of disc files. The authors distinguish three patterns of use, which they call scientific computing, transaction processing, and engineering/office applications. The method described is appropriate to the last of these, which is characterised by the need for large numbers of quite small files – and which therefore don't directly gain very much from disc arrays, as there is little requirement for long sequential read or write operations which must be completed quickly and can usefully be run in parallel over several disc drives.

DISC CACHES.

As a first step, we can try to keep all the wanted files in memory using a standard caching technique : an area of memory is reserved to hold copies of the disc sectors used, and managed so that the most recently used sectors are immediately available from the cache. (A disc cache is quite similar in principle to the more familiar memory cache, but stores disc sectors rather than memory words.) Small files can live completely within the cache; given total reliability, they need only be written to the disc when they are closed, and only then if the space they occupy is required for some more current file. For security in the real world, it is desirable to make sure that the disc copy of the file is identical with the cached version, so new information should be written to the disc as soon as possible, but as the size of the cache increases it will become less and less common to need to read information from the disc (after the first time, if the file exists before the programme starts). The disc cache can therefore be written to the disc as time becomes available, imposing very little load on the system.

What happens if the power fails ? Provided we have battery backup, nothing very dreadful. Generally, we might need to protect the cache from a variety of sorts of unfortunate accident, but we can do that fairly well even now. With improvements in reliability, there might be very little need to worry about such failures in future.

In that case, the disc really does become little more than an archive for important data which must be saved over a long period of time. We will hardly ever want to read material back from the disc; so we can concentrate on writing to the disc, and we can devise our system to write to the disc as efficiently as possible.

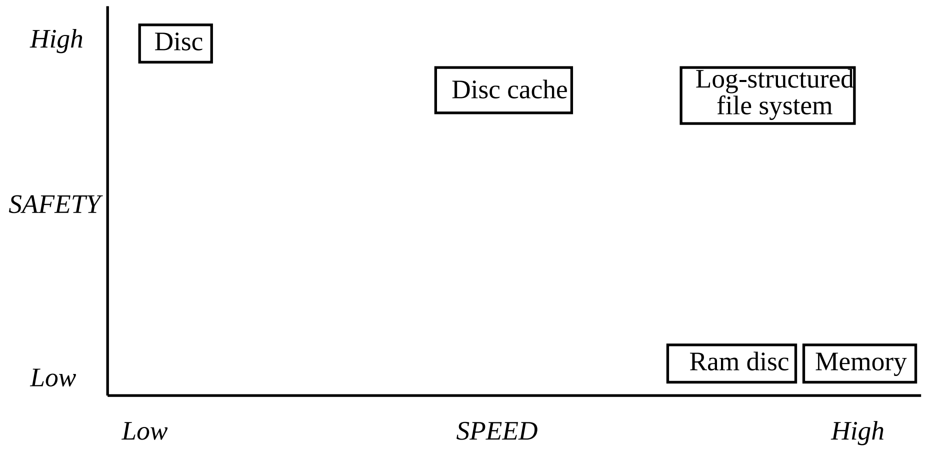
THE LOG-STRUCTURED FILE SYSTEM.

The most efficient way to write to a disc is sequentially – so why not simply write all filed material to the disc as it is sent ? We can worry about getting it back when it's necessary, and it doesn't matter much if that is rather inefficient. Obviously there's a need for a certain amount of housekeeping information about which file is which and how the bits are connected, but that can just be written sequentially too.

The authors of the paper analyse this proposal in some detail – and they find that they can design a method which is both maximally efficient for writing to the disc, and also efficient for reading back. They make a very plausible case. Is this the shape of things to come ?

COMPARING METHODS.

The graph below gives as very schematic picture of the relative performances of some of the methods we've discussed.



WHO NEEDS A FILE SYSTEM ?

Perhaps the log-structured system isn't the last word. Consider what it does. There are two components : it moves the "real" file system into memory, and it provides an ingenious and efficient way to back up the system to disc to guard against catastrophe. In other words, it provides a secure RAM disc; but it accepts without question that we want a file system. Do we ?

We suggest that we don't. If we're going to keep all the information in memory, why don't we organise it in terms of more useful data structures ? For years we've had to go to all sorts of lengths to take complicated data structures to pieces and then to write the pieces, and all manner of structural information, onto files in such a way that we could get the original data structures back again. If we believe that memory is really so reliable, then why not just keep the data structures ? Provided that someone will build us software resembling the log-structured file system software to keep a permanent record of the way the structure changes, then we'll be better off.

This proposal is an extension of the suggestion put forward in the chapter *FILES – FROM THE BEGINNING* that random access files should behave like arrays rather than like traditional files, the only difference being in the attribute of persistence. Some ideas of this sort are under discussion, particularly as increasing processor address spaces make the idea of persistent arrays "in memory" a possibility.

In practice, we are likely to want to retain at least one vestige of the file system – the directory, though perhaps in an advanced form. It's all very well saying that we can keep records of everything we've ever done in our 2^{64} bit address space, but how do we find it again ten years later ? No one is going to remember the literal binary addresses (we'd hope that no one will ever need to see one); we'll forget names, and tend to use them again quite by accident (which the system must certainly allow without necessarily deleting the earlier instance, or the names we use will have to become more and more unnatural as the years pass). We will therefore need a way of retrieving the items which will cope with names if we remember them, but also let us use specifications which describe the content of the objects.

REFERENCES.

SUP6 : A. Bartioli, S.J. Mullender, M. van der Valk : "Wide address spaces – exploring the design space", *Operating Systems Review* **27#1**, 11 (January 1993)

IMP26 : J. Ousterhout, F. Douglis : "Beating the I/O bottleneck : a case for log-structured file systems", *Operating Systems Review* **23#1**, 11 (January 1989)

IMP27 : M. Rosenblum, J.K. Ousterhout : "The design and implementation of a log-structured file system", *Operating Systems Review* **25#5**, 1 (Special issue, 1991).

QUESTIONS.

Is it sensible to think of the disc as an archive cache ?

Consider the hierarchy of caches, from registers to archive. Is it a sound general principle that it's a good idea to use faster memory to save some recent data from the next slower memory type ?
