PAPER *Special Section on Hardware/Software Support for High Performance Scientific and Engineering Computing*

Traditional File Systems versus DualFS: A Performance Comparison Approach*

Juan PIERNAS[†a)], **Toni CORTES**[††], ***and*** **José M. GARCÍA**[†], ***Nonmembers***

SUMMARY DualFS is a next-generation journaling file system which has the same consistency guaranties as traditional journaling file systems but better performance. This paper introduces three new enhancements which significantly improve DualFS performance during normal operation, and presents different experimental results which compare DualFS and other traditional file systems, namely, Ext2, Ext3, XFS, JFS, and ReiserFS. The experiments carried out prove, for the first time, that a new file system design based on separation of data and metadata can significantly improve file systems' performance without requiring several storage devices.

key words: *DualFS, data and meta-data separation, file systems' comparison, storage devices, journaling*

1. Introduction

High-performance computing environments rely on several hardware and software elements to fulfill their jobs. Although some environments require specialized elements, many of them use off-the-shelf elements which must have a good performance in order to accomplish computing jobs as quickly as possible.

A key element in these environments is the file system. File systems provide a friendly means to store final and partial results, from which it is even possible to resume a failed job because of a system crash, program error, or whatever other fail. Some computing jobs require a fast file system, others a quickly-recoverable file system, and other jobs require both features.

DualFS [1] is a new concept of journaling file system which is aimed at providing both good performance and fast consistency recovery. DualFS, like other file systems [2]–[6], uses a meta-data log approach for fast consistency recovery, but from a quite different point of view. Our new file system separates data blocks and meta-data blocks completely, and manages them in very different ways. Data blocks are placed on the *data device* (a disk partition) whereas meta-data blocks are placed on the *meta-data device* (another partition in the same disk). In order to guarantee a fast consistency recovery after a system crash, the meta-data device is treated as a log-structured file system [4], whereas the data device is divided into groups in order to organize data blocks (like other file systems do [3], [5], [7]–[9]).

Although the separation between data and meta-data is not a new idea, this paper introduces a new version of DualFS which proves, for the first time, that the separation can significantly improve file systems' performance without requiring several storage devices, unlike previous proposals [10], [11].

The new version of DualFS has several important improvements with respect to the previous one [1]. First of all, DualFS has been ported to the 2.4.19 Linux kernel. Secondly, we have greatly changed the implementation and removed some important bottlenecks. And finally, we have added three new enhancements which take advantage of the special structure of DualFS, and significantly increase its performance. These enhancements are: *meta-data prefetching*, *on-line meta-data relocation*, and *directory affinity*.

The resultant file system has been compared with Ext2 [7], Ext3 [9], XFS [5], [12], JFS [3], and ReiserFS [13], through an extensive set of benchmarks, and the experimental results achieved not only show that DualFS is the best file system in general, but also that it can remarkably boost the overall system performance in many cases.

2. DualFS

This section summarizes the main features of DualFS, and describe the three enhancements that we added to the new version of DualFS analyzed in this paper.

2.1 Design and Implementation

The key idea of our new file system is to manage data and meta-data in completely different ways, giving a special treatment to meta-data [1]. Meta-data blocks are located on the *meta-data device*, whereas data blocks are located on the *data device* (see Fig. 1).

Although separating meta-data blocks from data blocks is not a new idea, we will show that the separation, along with a special meta-data management, can significantly improve performance without requiring extra hardware. In our experiments, the data device and the meta-data device are two adjacent partitions on the same disk.

Manuscript received October 15, 2003.
Manuscript revised January 15, 2004.

[†]The authors are with Universidad de Murcia, Spain.
[††]The author is with Universitat Politecnica de Catalunya, Barcelona, Spain.

[*]This work has been supported by the Spanish Ministry of Science and Technology and the European Union (FEDER funds) under the TIC2000-1151-C07-03 and TIC2001-0995-C02-01 CICYT grants.

a) E-mail: piernas@ditec.um.es

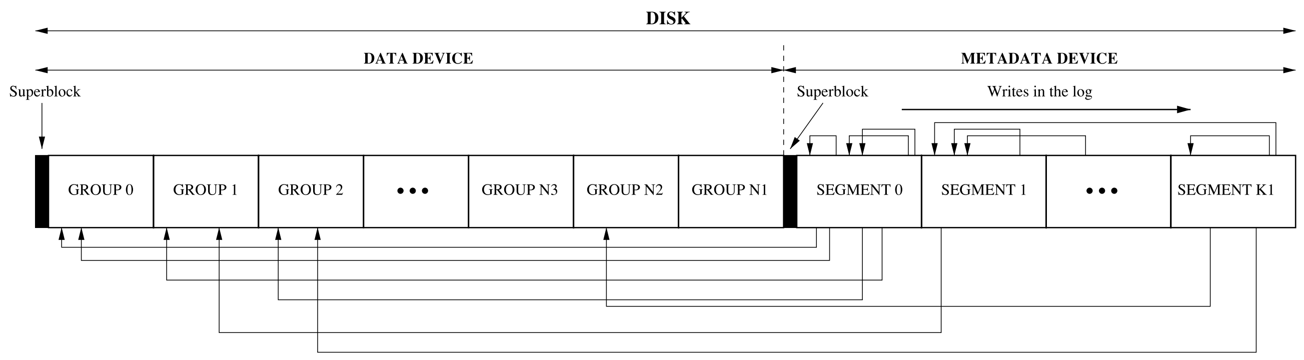


Fig. 1 DualFS overview. Arrows are pointers stored in segments of the meta-data device. These pointers are numbers of blocks in both the data device (data blocks) and the meta-data device (meta-data blocks).

Data Device

Data blocks of regular files are on the data device, where they are treated as many Unix file systems do. The data device only has data blocks, and uses the concept of *group of data blocks* (similar to the cylinder group concept) in order to organize data blocks (see Fig. 1). Grouping is performed in a per directory basis, i.e., data blocks of regular files in the same directory are in the same group (or in near groups if the corresponding group is full).

Meta-data Device

Meta-data blocks are in the meta-data device which is organized as a log-structured file system (see Fig. 1). It is important to clarify that as meta-data we understand all these items: i-nodes, indirect blocks, directory "data" blocks, and symbolic link "data" blocks. Obviously, bitmaps, superblock, etc., are also meta-data.

Our implementation of log-structured file system for meta-data is based on the BSD-LFS one [14]. However, it is important to note that, unlike BSD-LFS, our log does not contain data blocks, only meta-data ones. Another difference is the IFile. Our IFile implementation has two additional elements which manage the data device: the *data-block group descriptor table*, and the *data-block group bitmap table*. The former basically has a free data-block count and an i-node count per group, while the latter indicates which blocks are free and which are busy, in every group.

DualFS writes dirty meta-data blocks to the log every five seconds and, like other journaling systems, it uses the log after a system crash to recover the file system consistency quickly from the last checkpoint (checkpointing is performed every sixty seconds). Note, however, that one of the main differences between DualFS and current journaling file systems [2], [5], [6] is the fact that DualFS only has to write one copy of every meta-data block. This copy is written in the log, which is used by DualFS both to read and write meta-data blocks. In this way, DualFS avoids a time-expensive extra copy of meta-data blocks. Furthermore, since meta-data blocks are written only once and in big chunks in the log, the average write request size is greater in DualFS than in other file systems where meta-data blocks are spread. This causes less write requests and, hence, a greater write performance [1].

Finally, it is important to note that our file system is considered consistent when information about meta-data is correct. Like other approaches (JFS [3], SoftUpdates [15], XFS [5], and VxFS [6]), DualFS allows some loss of data in the event of a system crash.

Cleaner

Since our meta-data device is a log-structured file system, we need a segment cleaner for it. Our cleaner is started in two cases: (a) every 5 seconds, if the number of clean segments drops below a specific threshold, and (b) when we need a new clean segment, and all segments are dirty.

At the moment our attention is not on the cleaner, so we have implemented a simple one, based on Rosenblum's cleaner [4]. Despite its simplicity, this cleaner has a very small impact on DualFS performance [1].

2.2 Enhancements

Reading a regular file in DualFS is inefficient because data blocks and their related meta-data blocks are a long way from each other, and many long seeks are required.

In this section we present a solution to that problem, *meta-data prefetching*, a mechanism to improve that prefetching, *on-line meta-data relocation*, and a third mechanism to improve the overall performance of data operations, *directory affinity*.

Meta-data Prefetching

A solution to the read problem in DualFS is meta-data prefetching. If we are able to read a lot of meta-data blocks of a file in a single disk I/O operation, then disk heads will stay in the data zone for a long time. This will eliminate almost all the long seeks between the data device and the meta-data device, and it will produce bigger data read requests.

The meta-data prefetching implemented in DualFS is very simple: when a meta-data block is required, DualFS reads a group of consecutive meta-data blocks from disk, where the meta-data block required is. Prefetching is not performed when the meta-data block requested is already in memory. The idea is not to force an unnecessary disk I/O request, but to take advantage of a compulsory disk-head movement to the meta-data zone when a meta-data block must be read, and to prefetch some meta-data blocks then. Since all meta-data blocks prefetched are consecutive, we also take advantage of the built-in cache of the disk drive.

But, in order to be efficient, our simple prefetching requires some kind of meta-data locality. The DualFS meta-data device is a log where meta-data blocks are sequentially written in chunks called "partial segments". All meta-data blocks in a partial segment have been created or modified at the same time. Hence, some kind of relationship between them is expected. Moreover, many relationships between meta-data blocks are due to those meta-data blocks belonging to the same file (e.g., indirect blocks). This kind of temporal and spatial meta-data locality presents in the DualFS log is which makes our prefetching highly efficient.

Once we have decided when to prefetch, the next step is to decide which and how many meta-data blocks must be prefetched. This decision depends on the meta-data layout in the DualFS log, and this layout, in turn, depends on the meta-data write order.

In DualFS, meta-data blocks belonging to the same file are written to the log in the following order: "data" blocks (of directories and symbolic links), simple indirect blocks, double indirect blocks, and triple indirect blocks. Furthermore, "data" blocks are written in inverse logical order. Files are also written in inverse order: first, the last modified (or created) file, and last, the first modified (or created) file. Note that the important fact is not that the logical order is "inverse"; which is important is that the logical order is "known".

Taking into account all the above, we can see that the greater part of the prefetched meta-data blocks must be just before the required meta-data block. However, since some files can be read in an order slightly different to the one in which they were written, we must also prefetch some meta-data blocks which are located after the requested meta-data block.

Finally, note that, unlike other prefetching methods [16], [17], DualFS prefetching is I/O-time efficient, that is, it does not cause extra I/O requests which can in turn cause long seeks. Also note that our simple prefetching mechanism works because it takes advantage of the unique features of our meta-data device.

On-line Meta-data Relocation

The meta-data prefetching efficiency may deteriorate due to several reasons:

- files are read in an order which is very different from the order in which they were written.
- read patterns change over the course of the time.
- file-system aging.

An inefficient prefetching increases the number of meta-data read requests and, hence, the number of disk-head movements between the data zone and the meta-data zone. Moreover, it can become counterproductive because it can fill the buffer cache up with useless meta-data blocks. In order to avoid prefetching degradation (and to improve its performance in some cases), we have implemented an on-line meta-data relocation mechanism in DualFS which increases temporal and spatial meta-data locality.

The meta-data relocation works as follows: when a meta-data element (i-node, indirect block, directory block, etc.) is read, it is written in the log like any other just modified meta-data element. Note that it does not matter if the meta-data element was already in memory when it was read or if it was read from disk. Relocation is mainly performed in two cases:

- when reading a regular file (its indirect blocks are relocated).
- when reading a directory (its "data" and indirect blocks are relocated).

This constant relocation adds more meta-data writes. However, meta-data writes are very efficient in DualFS because they are performed sequentially and in big chunks. Therefore, it is expected that this relocation, even when very aggressive, does not increase the total I/O time significantly.

Since a file is usually read in its entirety [18], the meta-data relocation puts together all meta-data blocks of the file. The next time the file is read, all its meta-data blocks will be together, and the prefetching will be very efficient.

The meta-data relocation also puts together the meta-data blocks of different files read at the same time (i.e., the meta-data blocks of a directory and its regular files). If those files are also read later at the same time, and even in the same order, the prefetching will work very well. This assumption in the read order is made by many prefetching techniques [16], [17]. It is important to note that our meta-data relocation exposes the relationships between files by writing together the meta-data blocks of the files read at the same time. Unlike other prefetching methods [16], [17], this relationship is permanently recorded on disk, and it can be exploited by our prefetching mechanism after a system restart.

This kind of on-line meta-data relocation is in a sense similar to the work proposed by Jeanna N. Matthews *et al.* [19]. They take advantage of cached data in order to reduce cleaning costs. We write recently read and, hence, cached meta-data blocks to the log in order to improve meta-data locality and prefetching. They also propose a dynamic reorganization of data blocks to improve read performance. However, that reorganization is not an on-line one, and it is more complex than ours.

Finally, we must explain an implicit meta-data relocation which we have not mentioned yet. When a file is read, the *access time* field in its i-node must be updated. If several files are read at the same time, their i-nodes will also be updated at the same time, and written together in the log. In this way, when an i-node is read later, the other i-nodes in the same block will also be read. This implicit prefetching is very important since it exploits the temporal locality in the log, and can potentially reduce file open latencies, and the overall I/O time. Note that this i-node relocation takes place, without an explicit meta-data relocation.

This implicit meta-data relocation can have an effect similar to that achieved by the embedded i-nodes proposed by Ganger and Kaashoek [20]. In their proposal, i-nodes of files in the same directory are put together and stored inside the directory itself. Note, however, that they exploit the spatial locality, whereas we exploit both spatial and temporal localities.

Directory Affinity

In DualFS, like in Ext2 and Ext3, when a new directory is created it is assigned a data-block group in the data device. Data blocks of the regular files created in that directory are put in the group assigned to the directory. DualFS specifically selects the emptiest data-block group for the new directory.

Note that, when creating a directory tree, DualFS has to select a new group every time a directory is created. This can cause a change of group and, hence, a large seek. However, the new group for the new directory may be only slightly better than that of the parent directory. A better option is to remain in the parent directory's group, and to change only when the new group is good enough, according to a specific threshold. The latter is called *directory affinity* in DualFS, and can greatly improve DualFS performance.

Directory affinity in DualFS is somewhat similar to the "directory affinity" concept used by Anderson *et al.* [10], but with some important differences. For example, their directory affinity is a probability (the probability that a new directory is placed on the same server as its parent) while it is a threshold in our case.

3. Experimental Methodology

This section describes how we have evaluated the new version of DualFS. We have used both microbenchmarks and macrobenchmarks for different configurations, using the Linux kernel 2.4.19. We have compared DualFS against Ext2 [7], the default file system in Linux, and four journaling file systems: Ext3 [9], XFS [5], [12], JFS [3], and ReiserFS [13]. Ext2 is not a journaling file system, but it has been included because it is a widely-used and well-understood file system.

Bryant *et al.* [21] compared the above five file systems by using several benchmarks on three different systems, ranging in size from a single-user workstation to a 28-processor ccNUMA machine. However, there are some important differences between their work and ours. Firstly, we evaluate a next-generation journaling file system (DualFS). Secondly, we report results for some industry-standard benchmarks (SpecWeb99, and TPC-C). Thirdly, we use microbenchmarks which are able to clearly expose performance differences between file systems (in their single-user workstation system, for example, only one benchmark was able to show performance differences). And finally, we also report I/O time at disk level (this time is important because reflects the behavior of every file system as seen by the disk drive).

3.1 Microbenchmarks

We have designed seven microbenchmarks intended to discover strengths and weaknesses of every file system:

read-meta (r-m) find files larger than 2 KB in a directory tree.
read-data-meta (r-dm) read all the regular files in a directory tree.
write-meta (w-m) untar an archive which contains a directory tree with empty files.
write-data-meta (w-dm) the same as *w-m*, but with non-empty files.
read-write-meta (rw-m) copy a directory tree with empty files.
read-write-data-meta (rw-dm) the same as *rw-m*, but with non-empty files.
delete (del) delete a directory tree.

In all cases, the directory tree has 4 subdirectories, each with a copy of a clean Linux 2.4.19 source tree. In the "write-meta" and "read-write-meta" benchmarks, we have truncated to zero all regular files in the directory tree. In the "write-meta" and "write-data-meta" tests, the untarred archive is in a file system which is not being analyzed. All tests have been run 5 times for 1 and 4 processes. When there are 4 processes, each works on its own Linux source tree copy.

3.2 Macrobenchmarks

Next, we list the benchmarks we have performed to study the viability of our proposal. Note that we have chosen environments that are currently representative.

Kernel Compilation for 1 Process (KC-1P) resolve dependencies (*make dep*) and compile the Linux kernel 2.4.19, given a common configuration. Kernel and modules compilation phases are made for 1 process (*make bzImage*, and *make modules*).
Kernel Compilation for 4 Processes (KC-4P) the same as before, but just for 4 processes (*make -j4 bzImage*, and *make -j4 modules*).
SpecWeb99 (SW99) the SPECweb99 benchmark [22]. We have used two machines: a server, with the file system to be analyzed, and a client.

PostMark (PM) the PostMark benchmark, which models the workload seen by ISPs under heavy load [23]. We have run our experiments using version 1.5 of the benchmark. In our case, the benchmark initially creates 150,000 files with a size range of 512 bytes to 16 KB, spread across 150 subdirectories. Then, it performs 20,000 transactions with no bias toward any particular transaction type, and with a transaction block of 512 bytes.

TPCC-UVA (TPC-C) an implementation of the TPC-C benchmark [24]. Due to system limitations, we have only used 3 warehouses. The benchmark is run with an initial 30 minutes warm-up stage, and a subsequent measure time of 2 hours.

3.3 Tested Configurations

All benchmarks have been run for the six file systems showed below. Mount options have been selected following recommendations by Bryant *et al.* [21]:

Ext2, without any special mount option.
Ext3, with "`-o data=ordered`" mount option.
XFS, with "`-o logbufs=8,osyncisdsync`" options.
JFS, without any special mount option.
ReiserFS, with "`-o notail`" mount option.
DualFS, with meta-data prefetching, on-line meta-data relocation, and directory affinity.

Versions of Ext2, Ext3, and ReiserFS are those found in a standard Linux kernel 2.4.19. XFS version is 1.1, and JFS version is 1.1.1.

All file systems are on one IDE disk, and use a logical block size of 4 KB. DualFS uses two adjacent partitions, whereas the other file systems only use one disk partition. The DualFS meta-data device is always on the outer partition, since this partition is faster than the inner one [25], and its size is 10% of the total disk space. The inner partition is the data device.

Finally, DualFS has a prefetching size of 16 blocks, and a directory affinity of 10% (that is, a new-created directory is assigned the best group, instead of its parent directory's group, if the best group has, at least, 10% more free data blocks). Since the logical block size is 4 KB, the prefetching size is 64 KB, precisely the biggest disk I/O transfer size used by default by Linux.

3.4 System under Test

All tests are done in the same machine, whose configuration is shown in Table 1. In order to trace disk I/O activity, we have instrumented the operating system to record when a request starts and finishes.

4. Experimental Results

In order to better understand the different file systems, we show two performance results for each file system in each benchmark:

Table 1 System under test.

Processor	Two 450 Mhz Pentium III
Memory	256 MB, PC100 SDRAM
Disk	**Test disk**: One 4 GB 5,400 RPM Seagate ST-34310A IDE disk. **System disk** (operating system, and traces): One 4 GB 10,000 RPM FUJITSU MAC3045SC SCSI disk.
OS	Linux 2.4.19

- *Disk I/O time*: the total time taken for all disk I/O operations.
- *Performance*: the performance achieved by the file system in the benchmark.

The *performance* result is the *application time* except for the SpecWeb99 and TPC-C benchmarks. Since these macrobenchmarks take a fixed time specified by the benchmark itself, we will use the benchmark metrics (number of simultaneous connections for SpecWeb99, and tpmC for TPC-C) as throughput measurements.

We could give the application time only, but given that some macrobenchmarks (e.g, compilation) are CPU-bound in our system, we find I/O time more useful in those cases. The total I/O time can give us an idea of to what extent the storage system can be loaded. A file system that loads a disk less than other file systems makes it possible to increase the number of applications which perform disk operations concurrently.

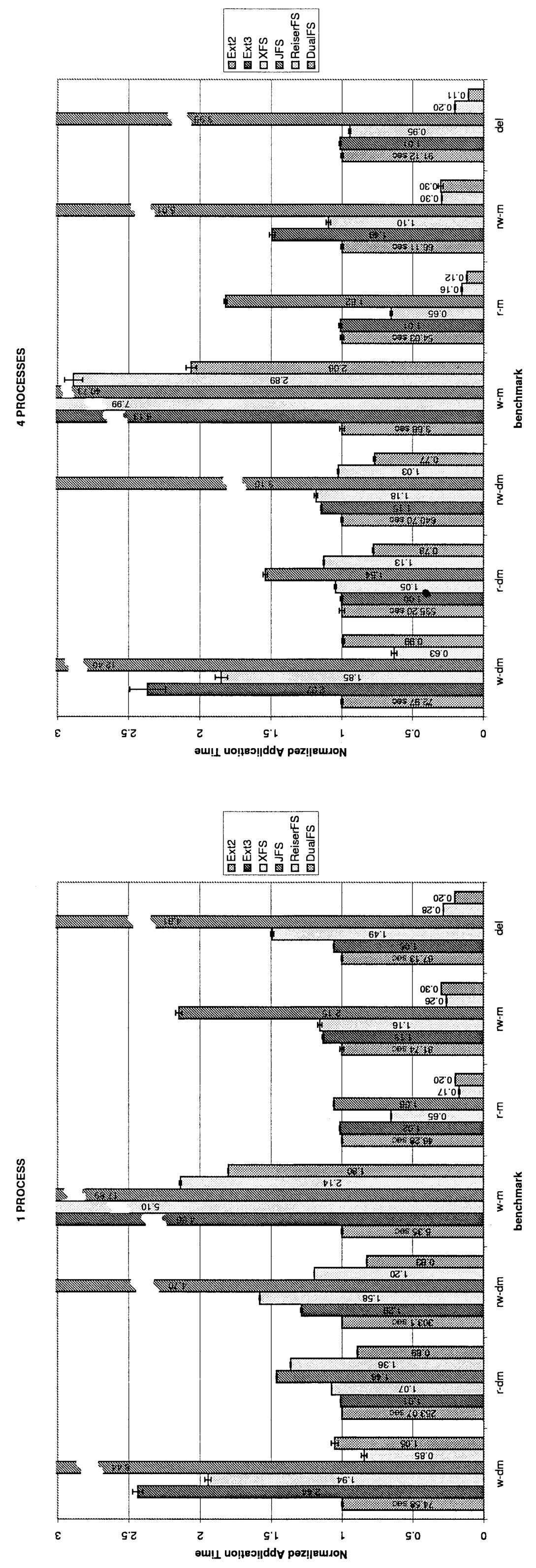
Figures below show the confidence intervals for the mean as error bars, for a 95% confidence level. The number inside each bar is the height of the bar, which is a value normalized with respect to Ext2. For Ext2 the height is always 1, so we have written the absolute value for Ext2 inside each Ext2 bar, which is useful for comparison purposes between figures.

The DualFS version evaluated in this section includes the three enhancements described previously. However, you can look at our previous work [26] if you want to know the impact of each enhancement (except directory affinity) on DualFS performance.

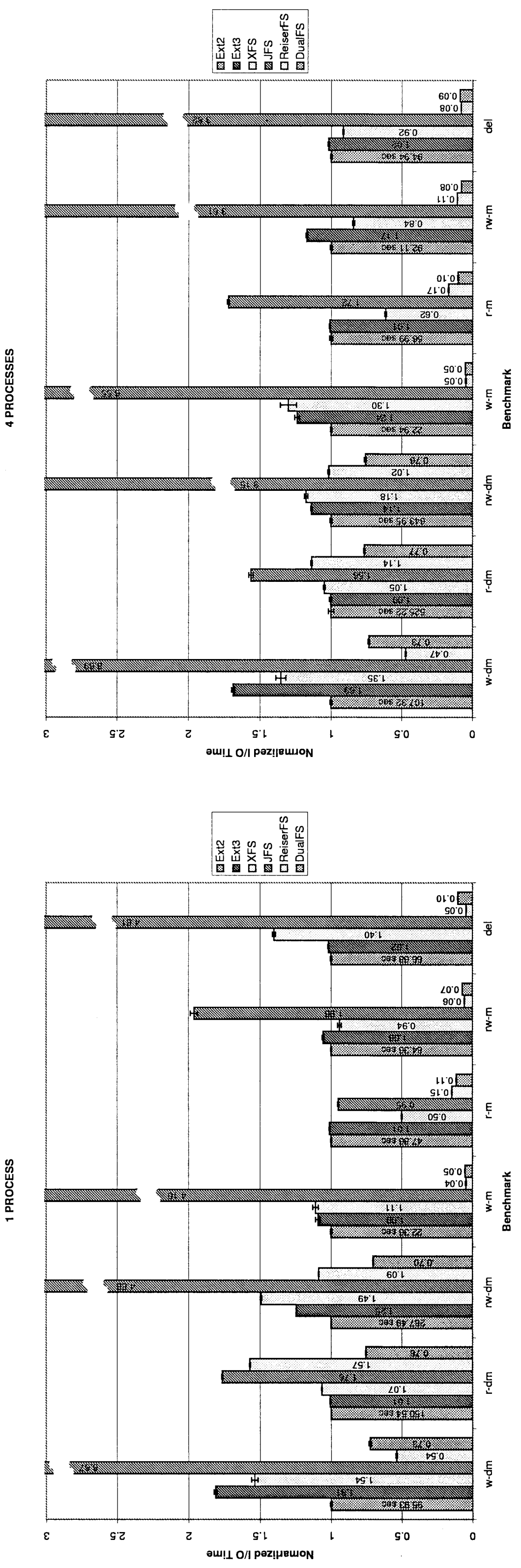
Finally, it is important to note that all benchmarks have been run with a cold file system cache (the computer is restarted after every test run).

4.1 Microbenchmarks

Figure 2 shows the microbenchmarks' results. A quick review shows that DualFS has the lowest disk I/O times and application times in general, in both write and read operations, and that JFS has the worst performance. Only ReiserFS is clearly better than DualFS in the *write-data-meta* benchmark, and it is even better when there are four processes. However, ReiserFS performance is very poor in the *read-data-meta* case. This is a serious problem for ReiserFS given that reads are a more frequent operation than writes



(a) Application time



(b) Disk I/O time

Fig. 2 Microbenchmarks results.

[18], [27]. In order to understand these results, we must explain some ReiserFS features.

ReiserFS does not use the block group or cylinder group concepts like the other file systems analyzed. Instead, ReiserFS uses an allocation algorithm which allocates blocks almost sequentially when the file system is empty. This allocation starts at the beginning of the file system, after the last block of the meta-data log. Since many blocks are allocated sequentially, they are also written sequentially. The other file systems, however, have data blocks spread across different groups which take up the entire file system, so writes are not as efficient as in ReiserFS. This explains the good performance of ReiserFS in the *write-data-meta* test.

The above also explains why ReiserFS is not so bad in the *read-data-meta* test when there are four processes. Since the directory tree read by the processes is small, and it is created in an empty file system, all its blocks are together at the beginning of the file system. This makes processes' read requests to cause small seeks when processes read the directory tree. The same does not occur in the other file systems, where the four processes cause long seeks because they read blocks which are in different groups spread across the entire disk.

Another interesting point in the *write-data-meta* benchmark is the performance of ReiserFS and DualFS with respect to Ext2. Although the disk I/O times of both file systems are much better than that of Ext2, the application times are not so good. This indicates that Ext2 achieves a better overlap between I/O and CPU operations because it neither has a journal nor forces a meta-data write order.

It is specially remarkable the good performance of DualFS in the *read-data-meta* benchmark, where DualFS is up to 35% faster than ReiserFS. When compared with previous versions of DualFS [1], we can see that this performance is mainly provided by the the meta-data prefetching and directory affinity mechanisms, which respectively reduce the number of long seeks between data and meta-data partitions, and between data-block groups within the data partition.

In all the metadata-only benchmarks but *write-meta*, the distinguished winners (regarding the application time) are ReiserFS and DualFS. And they are the absolute winners taking into account the disk I/O time. Also note that DualFS is the best when the number of processes is four.

The problem of the *write-meta* benchmark is that it is CPU-bound because all modified meta-data blocks fit in main memory. In this benchmark, Ext2 is very efficient whereas the journaling file systems are big consumers of CPU time in general. Even then, DualFS is the best of the five. Regarding the disk I/O time, ReiserFS and DualFS are the best, as expected, because they write meta-data blocks to disk sequentially.

In the *read-meta* case, ReiserFS and DualFS have the best performance because they read meta-data blocks which are very close. Ext2, Ext3, XFS, and JFS, however, have meta-data blocks spread across the storage device, which causes long disk-head movements. Note that the DualFS performance is even better when there are four processes.

Table 2 Results of the *r-m* test. The value in parenthesis is the confidence interval given as percentage of the mean.

	I/O Time (seconds)		
File System	1 process	4 processes	Increase (%)
Ext2	47.86 (0.35)	56.99 (1.13)	19.08
Ext3	48.45 (0.29)	57.59 (0.35)	18.86
XFS	24.00 (0.66)	35.13 (1.15)	46.38
JFS	45.49 (0.46)	98.10 (0.44)	115.65
ReiserFS	7.02 (1.44)	9.77 (2.32)	39.17
DualFS	5.46 (2.00)	5.90 (3.33)	8.06

This increase in DualFS performance, when the number of processes goes from one to four, is due to the meta-data prefetching. Indeed, prefetching makes DualFS scalable with the number of processes. Table 2 shows the I/O time for the six file systems studied, and for one and four processes. Whereas Ext2, Ext3, XFS, JFS, and ReiserFS significantly increase the total I/O time when the number of processes goes from one to four, DualFS increases the I/O time slightly.

For one process, the high meta-data locality in the DualFS log and the implicit prefetching performed by the disk drive (through the read-ahead mechanism) make the difference between DualFS, and Ext2, Ext3, JFS, and XFS. ReiserFS also takes advantage of the same disk read-ahead mechanism. However, that implicit prefetching performed by the disk drive is less effective if the number of processes is two or greater. When there are four processes, the disk heads constantly go from track to track because each process works with a different area of the meta-data device. When the disk drive reads a new track, the previous read track is evicted from the built-in disk cache, and its meta-data blocks are discarded before being requested by the process which caused the read of the track.

The explicit prefetching performed by DualFS solves the above problem by copying meta-data blocks from the built-in disk cache to the buffer cache in main memory before being evicted. Meta-data blocks can stay in the buffer cache for a long time, whereas meta-data blocks in the built-in cache will be evicted soon, when the disk drive reads another track.

Another remarkable point in the *read-meta* benchmark is the XFS performance. Although XFS has meta-data blocks spread across the storage device like Ext2 and Ext3, its performance is much better. We have analyzed XFS disk I/O traces and we have found out that XFS does not update the "atime" of directories by default. The lack of meta-data writes in XFS reduces the total I/O time because there are fewer disk operations, and because the average time of the read requests is smaller. JFS does not update the "atime" of directories either, but that does not significantly reduce its I/O time.

In the last benchmark, *del*, the XFS behavior is odd again. For one process, it has a very bad performance. However, the performance improves when there are four processes. The other file systems have the behavior we expect.

Finally, note the great performance of DualFS in the

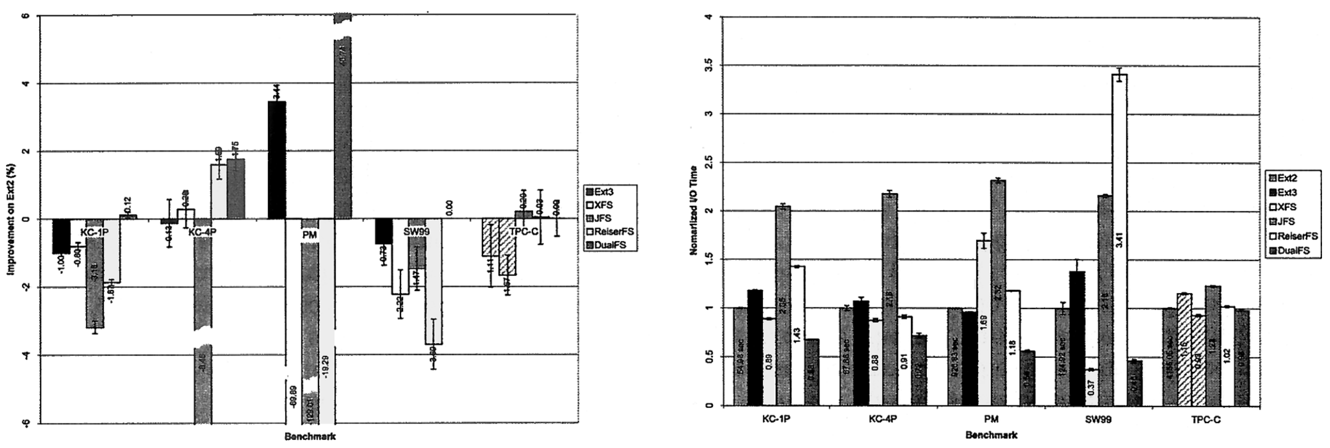


Fig. 3 Macrobenchmark results. The first figure shows the application throughput improvement achieved by each file system with respect to Ext2. The second figure shows the disk I/O time normalized with respect to Ext2. In the TPC-C benchmark, Ext3 and XFS bars are striped because they do not meet TPC-C benchmark requirements.

read-data-meta, and *read-meta* benchmarks despite the on-line meta-data relocation.

4.2 Macrobenchmarks

Results of macrobenchmarks are showed in Fig. 3. Since benchmark metrics are different, we have showed the relative application performance with respect to Ext2 for every file system.

As we can see, the only I/O-bound benchmark is PostMark (that is, benchmark results agree with I/O time results). The other four benchmarks are CPU-bound in our system, and all file systems consequently have a similar performance. Nevertheless, DualFS is usually a little better than the other file systems. The reason to include these CPU-bound benchmarks is that they are very common for system evaluations.

From the disk I/O time point of view, DualFS has the best throughput. Only XFS is better than DualFS in the SpecWeb99 and TPC-C benchmarks. However, we must respectively take into account that XFS does not update the access time of directory i-nodes, and nor does it meet the TPC-C benchmark requirements (specifically, it does not meet the response time constraints for new-order and order-status transactions).

In order to explain this superiority of DualFS over the other file systems, we have analyzed the disk I/O traces obtained, and we have found out that performance differences between file systems are mainly due to writes. There are a lot of write operations in these tests, and DualFS is the file system which better carries out them. For JFS, however, these performance differences are also due to reads, which take longer than in the other file systems.

Internet System Providers should pay special attention to the results achieved by DualFS in the PostMark benchmark. In this test, DualFS achieves 60% more transactions per second than Ext2 and Ext3, twice as many transactions per second as ReiserFS, almost three times as many transactions per second as XFS, and four as many transactions per second as JFS. Although there are specific file systems for Internet workloads [28], note that these results are achieved by a general-purpose file system, DualFS.

5. Conclusions

Improving file system performance is important for a wide variety of systems, from general purpose systems to more specialized high-performance systems. Many high-performance systems, for example, rely on off-the-shelf file systems to store final and partial results, and to resume failed computation.

In this paper we have introduced a new version of DualFS which proves, for the first time, that a new journaling file-system design based on data and meta-data separation, and special meta-data management, is not only possible but also desirable.

Through an extensive set of micro- and macrobenchmarks, we have evaluated six different journaling and non-journaling file systems (Ext2, Ext3, XFS, JFS, ReiserFS, and DualFS), and the experimental results obtained not only have shown that DualFS has the lowest I/O time in general, but also that it can significantly boost the overall file system performance for many workloads.

We feel, however, that the new design of DualFS allows further improvements which can increase DualFS performance even more. Currently, we are working on extracting read access pattern information from meta-data blocks written by the relocation mechanism in order to prefetch entire regular files.

Availability

http://www.ditec.um.es/˜piernas/dualfs.

References

[1] J. Piernas, T. Cortes, and J.M. García, “DualFS: A new journaling file system without meta-data duplication,” Proc. 16th Annual

ACM International Conference on Supercomputing, pp.137–146, June 2002.
[2] S. Chutani, O.T. Anderson, M.L. Kazar, B.W. Leverett, W.A. Mason, and R. Sidebotham, “The Episode file system,” Proc. Winter USENIX Conf., pp.43–60, Jan. 1992.
[3] “JFS for Linux,” http://oss.software.ibm.com/jfs, 2003.
[4] M. Rosenblum and J. Ousterhout, “The design and implementation of a log-structured file system,” ACM Trans. Computer Systems, vol.10, no.1, pp.26–52, Feb. 1992.
[5] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck, “Scalability in the XFS file system,” Proc. USENIX Annual Tech. Conf., pp.1–14, Jan. 1996.
[6] “The VERITAS File System,” http://www.veritas.com
[7] R. Card, T. Ts’o, and S. Tweedie, “Design and implementation of the second extended filesystem,” Proc. First Dutch International Symposium on Linux, Dec. 1994.
[8] M. McKusick, M. Joy, S. Leffler, and R. Fabry, “A fast file system for UNIX,” ACM Trans. Computer Systems, vol.2, no.3, pp.181–197, Aug. 1984.
[9] S. Tweedie, “Journaling the Linux ext2fs filesystem,” Linux-Expo’98, pp.1–8, 1998.
[10] D.C. Anderson, J.S. Chase, and A.M. Vahdat, “Interposed request routing for scalable network storage,” Proc. Fourth USENIX OSDI, pp.259–272, Oct. 2000.
[11] K. Muller and J. Pasquale, “A high performance multi-structured file system design,” Proc. 13th ACM SOSP, pp.56–67, Oct. 1991.
[12] “Linux XFS,” http://linux-xfs.sgi.com/projects/xfs
[13] “ReiserFS,” http://www.namesys.com
[14] M. Seltzer, K. Bostic, M.K. McKusick, and C. Staelin, “An implementation of a log-structured file system for UNIX,” Proc. Winter USENIX Conf., pp.307–326, Jan. 1993.
[15] M.K. McKusick and G.R. Ganger, “Soft updates: A technique for eliminating most synchronous writes in the fast filesystem,” Proc. 1999 USENIX Annual Tech. Conf., pp.1–17, June 1999.
[16] T.M. Kroeger and D.D.E. Long, “Design and implementation of a predictive file prefetching algorithm,” Proc. USENIX Annual Tech. Conf., pp.319–328, June 2001.
[17] H. Lei and D. Duchamp, “An analytical approach to file prefetching,” Proc. 1997 USENIX Annual Tech. Conf., pp.275–288, 1997.
[18] M.G. Baker, J.H. Hartman, M.D. Kupfer, K.W. Shirriff, and J.K. Ousterhout, “Measurements of a distributed file system,” Proc. 13th ACM SOSP, pp.198–212, 1991.
[19] J.N. Matthews, D. Roselli, A.M. Costello, R.Y. Wang, and T.E. Anderson, “Improving the performance of log-structured file systems with adaptive methods,” Proc. ACM SOSP Conference, pp.238–251, Oct. 1997.
[20] G.R. Ganger and M.F. Kaashoek, “Embedded inodes and explicit grouping: Exploiting disk bandwidth for small files,” Proc. USENIX Annual Tech. Conf., pp.1–17, Jan. 1997.
[21] R. Bryant, R. Forester, and J. Hawkes, “Filesystem performance and scalability in Linux 2.4.17,” Proc. FREENIX Track: USENIX Annual Tech. Conf., pp.259–274, June 2002.
[22] “SPECweb99 Benchmark,” http://www.spec.org
[23] J. Katcher, “PostMark: A new file system benchmark,” TR3022, Network Appliance Inc., Oct. 1997.
[24] “TPCC-UVA,” http://www.infor.uva.es/˜diego/tpcc.html
[25] J. Wang and Y. Hu, “PROFS–performance-oriented data reorganization for log-structured file system on multi-zone disks,” Proc. Ninth MASCOTS International Symposium, pp.285–293, Aug. 2001.
[26] J. Piernas, T. Cortes, and J.M. García, “Improving the performance of new-generation journaling file systems through meta-data prefetching and on-line relocation,” Technical Report UM-DITEC-2003-5, Sept. 2002.
[27] W. Vogels, “File system usage in Windows NT 4.0,” ACM SIGOPS Operating Systems Review, vol.34, no.5, pp.93–109, Dec. 1999.
[28] E. Shriver, E. Gabber, L. Huang, and C. Stein, “Storage management for web proxies,” Proc. 2001 USENIX Annual Tech. Conf., pp.203–216, June 2001.

Juan Piernas is an Assistant Professor at Universidad de Murcia (Spain) since 1995, and a PhD candidate in Computer Science. He is writing its PhD dissertation about file systems’ design and implementation. His main interests are parallel I/O, file systems, and operating systems.

Toni Cortes is an Associate Professor at Universitat Politecnica de Catalunya (Barcelona – Spain) since 1999. He is currently the coordinator of the single-system image technical area in the IEEE Task Force on Cluster Computing (TFCC), vicedean for international affairs at the Barcelona School of Informatics, and manager at the CEPB-IBM research institute.

José M. García is an Associate Professor at Universidad de Murcia (Spain) since 1994. At present, he is the Director of the Computer Engineering Department, and also the Head of the Research Group on Parallel Computing Architecture. He has published over 50 refereed papers in different journals and conferences in fields such as Multiprocessor Systems, Grid Computing, etc.