DualFS: a New Journaling File System without Meta-Data Duplication *

Juan Piernas
Universidad de Murcia
piernas@ditec.um.es

Toni Cortes
U. Politècnica de Catalunya
toni@ac.upc.es

José M. García
Universidad de Murcia
jmgarcia@ditec.um.es

ABSTRACT

In this paper we introduce DualFS, a new high performance journaling file system that puts data and meta-data on different devices (usually, two partitions on the same disk or on different disks), and manages them in very different ways. Unlike other journaling file systems, DualFS has only one copy of every meta-data block. This copy is in the *meta-data device*, a log which is used by DualFS both to read and to write meta-data blocks. By avoiding a time-expensive extra copy of meta-data blocks, DualFS can achieve a good performance as compared to other journaling file systems. Indeed, we have implemented a DualFS prototype, which has been evaluated with microbenchmarks and macrobenchmarks, and we have found that DualFS greatly reduces the total I/O time taken by the file system in most cases (up to 97%), whereas it slightly increases the total I/O time only in a few and limited cases.

Categories and Subject Descriptors

D.4.3 [**Operating Systems**]: File Systems Management—*directory structures, file organization*; D.4.2 [**Operating Systems**]: Storage Management—*secondary storage*; E.5 [**Data**]: Files—*backup/recovery, organization/structure*

General Terms

Design, Performance, Reliability

Keywords

DualFS, journaling file system, meta-data management

1. INTRODUCTION

Meta-data management is one of the most important issues to be taken into account in the design of a file system. This management is especially important when the file system has to be recovered after a system crash, because it must be possible to bring the file system to a consistent state again. In order to guarantee this, file systems have traditionally written meta-data in a synchronous way, and have used tools (like `fsck` [11]) which scan the entire disk, after a crash, to solve potential inconsistencies.

The problem with `fsck`-like tools is that they can take a long time to scan an entire disk due to the large capacity of the current disks. In recent years, several solutions have been proposed [4, 18, 19, 26, 29] that keep some kind of log of the last meta-data updates, what allows us to recover the file system consistency quickly by analyzing only that log.

The synchronous write problem has also been studied at length [5, 6, 23]. Synchronous writes are used to enforce a specific order among meta-data writes. However, they downgrade the file system performance significantly, since they normally cause small I/O transfers at the speed of the underlying device. They can also cause a great bottleneck in many cases, specially in those cases which are sensitive to meta-data performance [18, 28].

To solve the aforementioned problems, current file systems treat data and meta-data somewhat differently while they are, in fact, completely different. Note, for example, that meta-data is the important part for file-system consistency while data is not as important [13, 26, 29].

The objective of this paper is to introduce a new high performance journaling file system, DualFS. This new file system separates data and meta-data completely and places them on different devices (possibly two partitions on the same disk). Once completely separated, data will be treated as regular Unix systems do [12], while meta-data will be treated as log-structured file system [19]. Like other journaling systems, DualFS uses the meta-data log to recover the file system consistency quickly after a system crash. However, one of the main differences between DualFS and other journaling file systems [4, 26, 29] is the fact that DualFS has only one copy of every meta-data block. This copy is in the log, which is used by DualFS both to read and to write meta-data blocks. Avoiding a time-expensive extra copy of meta-data blocks, DualFS can achieve a better performance than other journaling systems.

We will also present a working prototype in this paper, and we will show that this new organization, besides offering many new design opportunities, reduces the I/O time significantly in most of the cases.

*This work has been supported by the Spanish Ministry of Education (CICYT) under the TIC2000-1151-C07-03 and TIC2001-0995-C02-01 grants.

ICS'02, June 22-26, 2002, New York, New York, USA.

2. RELATED WORK

Multi-structured file systems [14] improve performance by separating data and meta-data structures, and placing them on independent and isolated disk arrays. Anderson *et al.* [1] use a similar approach to implement a new architecture for scalable network-attached storage. DualFS also separates data and meta-data. However, one important difference between these previous works and our new file system is the fact that DualFS does not need extra hardware in order to improve file system performance.

DualFS also attacks the meta-data consistency problem. One approach to solve this problem, without synchronously writing meta-data, is Soft Updates [13]. The idea behind Soft Updates is to have a more explicit control over dependencies among meta-data structures (i-nodes, indirect blocks, directory blocks, etc.) to know which meta-data updates depend on the update of a specific meta-data. This control of dependencies allows us to write meta-data updates in the right order. Granularity of this control is per individual meta-data structure, and not per block, in order to avoid cyclic dependencies. A block with modified meta-data can be written at any moment, provided that any update in the block which has pending dependencies is firstly "undone" temporarily.

Journaling file systems [26, 29] are another solution to the meta-data update problem. Basically, a journaling file system is an FFS (or another kind of file system) with an auxiliary log which records all meta-data operations. Therefore, meta-data operations take place both in the "normal" file system and in the log. This involves meta-data elements being duplicated. The file system enforces a *write-ahead logging*, a technique which ensures that the log is written to disk before any buffers containing data modified by the corresponding operations. If the system crashes, the log is replayed to recover the file system consistency. Journaling systems always perform additional I/O operations (the log writes) to maintain ordering information. Nevertheless, these additional writes can be done efficiently because they are sequential.

There are several differences between Soft Updates and journaling file systems [23]. Some of them are also differences between Soft Updates and DualFS. First, a journaling file system guarantees atomicity of updates. Since a journaling system records a logical operation, such as rename or create, it will always recover to either the pre-operation or post-operation state. Soft Updates, however, can recover to a state where both old and new names persist after a crash. Second, Soft Updates guarantees that the file system can be restarted without any file system recovery. At such a time, file system integrity is assured, but freed blocks and i-nodes may not yet be marked as free. A background process restores the file system to an accurate state with respect to free blocks and i-nodes. A journaling system requires a short recovery phase after system restart. Finally, a journaling file system duplicates meta-data writes using more time and space than actually needed. Soft Updates does not duplicate meta-data blocks.

Log-structured file systems (LFS) [19, 21] are another approach that solves both the synchronous meta-data problem and the small-write problem. Like Soft Updates, log-structured file systems ensure that blocks are written to disk in a specific order. Like journaling, they take advantage of sequential writes and log-based recovery. One difference between log-structured file systems and journaling file systems is that the former write both data blocks and meta-data blocks in the log, whereas the latter only write meta-data blocks. The main drawback of an LFS is *the cleaner*, a process which cleans segments and which may seriously downgrade the file system performance in many cases [22, 30].

Finally, there exist approaches that implement some kind of stable storage. The Rio file cache [3] makes ordinary memory safe for persistent storage, what allows a file system to avoid synchronous writes and guarantee the file system consistency at the same time. The RAPID-Cache [8] uses a disk to effectively increase the size of the write cache of another disk, improving the write performance of the second disk. Although DualFS pursues similar objectives, we think that DualFS is complementary to these approaches.

3. DUALFS

The main idea of this new file system is to manage data and meta-data in completely different ways. Each type of blocks, the meta-data and data ones, will be located on different devices, the *meta-data device*, and the *data device*, respectively. However, for the sake of security, there could be copies of some meta-data blocks on the data device. On account of that, the current prototype has a superblock copy on the data device.

3.1 Data Device

Data blocks of regular files are on the data device. The data device uses the concept of group of data blocks (similar to the cylinder group concept) in order to organize data blocks. In this way, we can bring together all data blocks of a file, even if the file grows sometime later.

The data device works as follows. When a directory is created, it is associated to a data block group, specifically to the emptiest data block group. Then, all regular files created in that directory place their data blocks in that group.

From the file-system point of view, data blocks are not important for consistency, so they are not written synchronously and do not receive any special treatment, as meta-data does [4, 23, 26, 29]. However, they must be taken into account for security reasons. When a new data block is added to a file, it must be written to disk before writing meta-data blocks related to that data block. Missing out this requirement would not actually damage the consistency, but it could potentially lead to a file containing a previous file's contents after crash recovery, which is a security risk. DualFS meets this requirement.

3.2 Meta-data Device

Meta-data is organized as a log-structured file system, that is, there is only one log where meta-data is written sequentially. It is important to clarify that by meta-data we understand all these items: i-nodes, indirect blocks, directory "data" blocks, and symbolic links (sometimes they need extra blocks which are treated as meta-data). Obviously, bitmaps, superblock copies, etc., are also meta-data elements.

DualFS can be seen as an evolution of a journaling file system, but with two important differences. On the one hand, meta-data blocks do not have to be written twice (one in the file system, one in the log), because there is only one copy of every meta-data element. On the other hand, data and meta-data blocks are not mixed; they can be on the

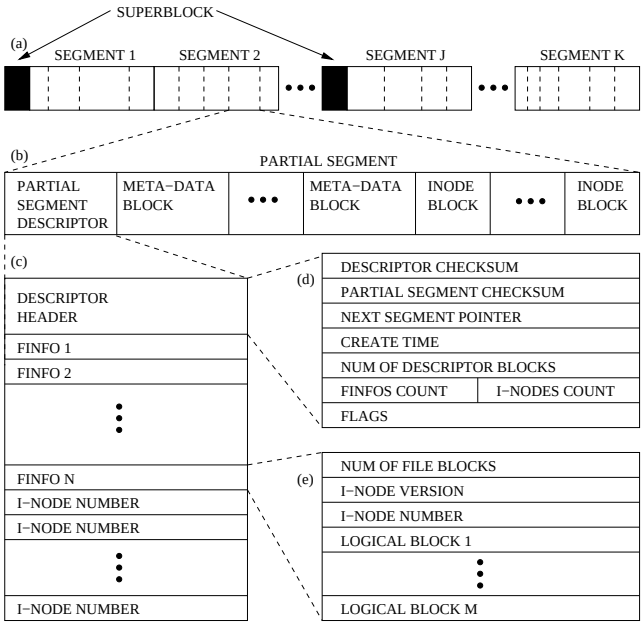

Figure 1: Structure of the meta-data device

same disk, in two different partitions, but they can be in two separate disks too. We hope that these two differences improve the DualFS performance for many workloads.

DualFS is also different to Soft Updates, another mechanism described in Section 2. Soft Updates can not sequentially write meta-data updates in big chunks, because there is nothing similar to a log. Hence, meta-data updates will be more efficient in DualFS than in Soft Updates.

Since data and meta-data blocks are separate, we suppose that a workload which will not take advantage of the DualFS features will be a read-only one, when both data and meta-data are on the same disk. In that case, data blocks and their related meta-data blocks will be a long way from each other, and DualFS will cause long seeks. Nevertheless, if each type of information is on different disks (data blocks on one disk, and meta-data blocks on another one) the above may not be true, as we will see in 5.3. The idea of using another disk for meta-data is not new; some journaling file systems allow us to put the log on a specific disk [26] in order to improve the file system performance. Journaling systems can exploit the parallelism offered by the extra disk only for meta-data writes. However, DualFS can exploit the parallelism between the two devices for both meta-data reads and meta-data writes.

On the other hand, there are a lot of large applications (such as those in Section 4.2) which are not read-only. In those applications, the separation between data and meta-data will be an advantage.

Implementation

Our implementation of a log-structured file system for meta-data is based on the BSD-LFS one [21]. However, it is important to note that, unlike BSD-LFS, our log does not contain data blocks, only meta-data ones.

The meta-data device is divided into pieces of equal size called ***segments*** (see Figure 1.a). Each segment is usually 1 MB or 2 MB in size.

Meta-data blocks are written in variable-sized chunks called ***partial segments*** (see Figure 1.b). Partial segments can be as large as a segment, although a segment often accommodates several partial segments which are written asynchronously. There is always one, and only one, active partial segment where the last modified meta-data blocks are written to. The active partial segment can be up to 5 seconds in main memory. This allows multiple meta-data updates to be batched into a single log write, and increases the efficiency of the log with respect to the underlying device [19]. This also means that, on average, we might lose the last 2.5 seconds of meta-data activity.

A partial segment is a *transaction unit*, and must be entirely written to disk to be valid. If a system crash occurs when a partial segment is being written, that partial segment will be dismissed during the file system recovery. Actually, a partial segment is a *compound transaction* because it is the result of several file system operations. Each file system operation is a transaction which is made up of the meta-data elements involved in that operation.

The structure of a partial segment is similar to that of BSD-LFS. The first element of the structure is the ***partial segment descriptor*** which is made up of a descriptor header, *finfo* structures, and i-node numbers (see Figure 1.c). All this information in the partial segment descriptor is used both for recovering a crashed file system rapidly, and for cleaning segments.

The *descriptor header* has information about the partial segment, and its structure is shown in Figure 1.d. Since a partial segment is a transaction unit, it is important to know when a partial segment on disk is valid. In order to know that, the descriptor header has two checksum fields, one for the entire descriptor (*descriptor checksum*), and one for the entire partial segment (*partial segment checksum*) – actually, the checksum of the entire partial segment is computed from the first 4 bytes of every partial segment block, since checksum computation is very CPU consuming.

Finfo structures are next. There is a finfo structure for every file (regular or not) that has one meta-data block (or more) in the partial segment. Fields which make up a finfo are shown in Figure 1.e. Logical blocks of a file are numbered as in BSD-LFS: data blocks have positive numbers, and indirect blocks have negative numbers assigned according to a specific numbering [21].

Finally, there is a *list of i-node numbers* (the numbers of the i-nodes in the partial segment). There are at least as many i-node numbers as finfo structures.

After the partial segment descriptor, there are meta-data blocks which belong to several files: indirect blocks, "data" blocks of directories, and "data" blocks of symbolic links.

At the end of the partial segment, there are several blocks which contain i-nodes. Every block contains as many non-consecutive i-nodes as it is able to, i.e., the number of blocks is the minimum to contain all the i-nodes in the partial segment.

Since i-nodes have not a fixed location on disk, we need an "i-node map", i.e., a structure that, given an i-node, returns the location of the i-node on disk. In our case, the i-node map is part of a special file, the ***IFile*** (see Figure 2). The structure of our IFile is similar to the one of the BSD-LFS IFile, but it has two additional elements: the DGDT, and the DGBT.

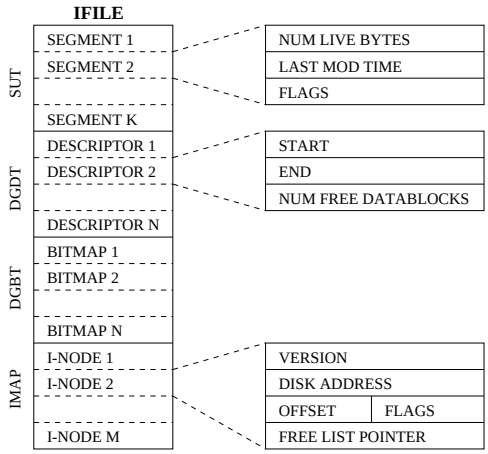

Figure 2: Structure of the IFile

The *segment usage table* (SUT) contains information about the amount of live bytes in each segment, when it was last modified, and some flags (the segment has a superblock copy, etc.). This information is used for cleaning and for selecting a new clean segment when it is needed.

The *data-block group descriptor table* (DGDT) has a role similar to the block group descriptor table of Ext2. As we do not have i-nodes (or other meta-data elements) on the data device, our data-block group descriptors do not contain information about the number of free i-nodes, or the number of directories, in each group. The only relevant information they have is the free data-block count.

After the DGDT, comes the *data-block group bitmap table*. We have a bitmap block for every data-block group, as Ext2 does. Every bitmap indicates which blocks are free and which are busy, in every group.

The last structure of the IFile is the *i-node map* (IMap). It comes at the end because it can grow; unlike other file systems, the number of i-nodes in DualFS can increase.

Cleaner

Like log-structured file systems, we need a segment cleaner. A segment may contain information still in use (the "live bytes"), but also obsolete information (information superseded by other information in a different segment, or information that is no longer valid). A segment that contains live bytes is a dirty segment. Otherwise, it is a clean segment. When the number of clean segments is small, the cleaner can collect the live bytes of several dirty segments and write them in a new segment. Hence, dirty segments will become clean ones. Obviously, if all dirty segments are full of live bytes, cleaning will be senseless.

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.

It is important to note that the number of meta-data blocks is usually much smaller than the number of data blocks in any file system. If our meta-data device is large enough, there will be always a lot of clean segments or a lot of dirty segments with few live bytes. Therefore, either our cleaner will be hardly required to work, or it will clean segments quickly. That is, our cleaner overhead will be small.

At the moment, our attention is not on the cleaner, so we have implemented a simple one, based on Rosenblum's cleaner [19]. The threshold we have chosen is also very simple (it is a fixed number), though it should depend on both the meta-data device size and the workload in a production file system.

Perhaps, our cleaner is not the best one, but it will allow us to obtain a conservative estimation of the impact of the cleaner on the DualFS performance.

3.3 Recovering a Failed File System

Our file system is considered consistent when information about meta-data is correct. Like other approaches [13, 26, 29], some loss of data is allowed in the event of a system crash.

Since our meta-data device is organized as a log-structured file system, we can recover the file system consistency very quickly from the last checkpoint. We write a checkpoint when the IFile is written. The superblock contains a reference to the last checkpoint.

Basically, recovering a DualFS file system means recovering its IFile. The IFile is only written to disk at each checkpoint, so its content may be lost at a system crash. During recovery, each partial segment is analyzed to update the IFile. Note that recovering does not involve redoing or undoing any meta-data operation, only updating information of the IFile. We scan the log from the last checkpoint to the last valid partial segment. Partial segments are timestamped and checksummed, so that the recovering process can easily detect when the end of the log is reached. The length of time for recovery is proportional to the inter-checkpoint interval.

4. EXPERIMENT METHODOLOGY

In this section we describe how we have evaluated the prototype of the new file system. We have used both microbenchmarks and macrobenchmarks for different configurations. DualFS has been compared against Ext2 [2], the default file system in Linux, and Ext3 [27], a journaling file system derived from Ext2, using the Linux kernel 2.2.19.

We have compared DualFS with Ext2 because Ext2 is the default file system in Linux, and because it is an FFS-like file system [12], a very common kind of file system in the Unix world. One important difference between Ext2 and other FFS-like file systems is that it does not write modified meta-data elements synchronously. Instead, it marks meta-data elements to be written in 5 seconds. Besides, meta-data elements involved in a file system operation are modified, and marked to be written, in a specific order. In this way, Ext2 can guarantee the consistency recovery after a system crash, without significantly damaging the performance.

Ext3 allows us to compare DualFS to a file system with similar consistency guarantees. Ext3 provides different consistency levels through mount options. In our tests, the mount option used has been "`-o data=ordered`", which provides Ext3 with a behavior similar to the DualFS one. With this option, Ext3 only journals meta-data changes, but flushes data updates to disk before any transactions commit.

Recently, other journaling file systems for Linux have emerged, such as XFS by SGI [24], and JFS by IBM [9]. We have not compared DualFS against these journaling file systems because they are not stable enough (version 1.0.4 of JFS hangs the computer in some of our microbenchmarks),

or because there is no version for the Linux kernel which we have used (XFS versions only exist for the Linux 2.4 series). We have also been unable to compare DualFS against Soft Updates, as there is no version of Soft Updates for Linux.

4.1 Microbenchmarks

Microbenchmarks are intended to discover strengths and weaknesses of DualFS. We have designed six benchmarks:

read-meta (r-m) find files larger than 2 KB in a directory tree.

read-data-meta (r-dm) search a pattern in all the regular files of a directory tree.

write-meta (w-m) create a directory tree with empty files from another tree used as a pattern.

write-data-meta (w-dm) create a directory tree from another one used as a pattern.

read-write-meta (rw-m) create a directory tree with empty files from another tree that is used as a pattern and is located in the same file system being analyzed.

read-write-data-meta (rw-dm) the same as *rw-m* but with non-empty files.

In all cases, the directory tree used as a pattern is a clean Linux 2.2.19 source tree. In the "write-meta" and "write-data-meta" tests, that directory is in a file system on a device which is not being analyzed.

All tests have been run 10 times for 1 and 4 processes. For 4 processes, job done by 1 process is divided into 4 roughly equal parts, each one for one process.

Finally, we have also used the *read-write-data-meta* test to obtain the CPU time consumed by each file system.

4.2 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 2.2.19, given a specific configuration. Kernel and modules compilation phases are done for 1 process (*make bzImage*, and *make modules*).

Kernel Compilation for 8 Processes (KC-8P) the same as before, but for 8 processes (*make -j8 bzImage*, and *make -j8 modules*).

Video Compression (VC) compress a video stream frame by frame. The video stream has 400 frames. Every frame is made up of three files. There is only one directory for all frame files. Each frame is compressed into one file. All output files are written in another directory. See [16] for further details.

Specweb99 (SW99) the SPECweb99 benchmark [25]. We have used two machines: a server, with the file system to be analyzed, and a client. Network is a FastEthernet LAN.

PostMark (PM) the PostMark benchmark, which was designed by Jeffrey Katcher to model the workload seen by Internet Service Providers under heavy load [10]. We have run our experiments using version 1.5 of the benchmark. With our configuration, 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.

4.3 Tested Configurations

Microbenchmarks and macrobenchmarks have been run for six configurations:

DualFS-1d DualFS on one disk with two partitions. The inner partition is the data device. The outer partition is the meta-data device.

DualFS-2d DualFS on two disks, each one with two partitions. The two inner partitions make up a software RAID 0, which is the data device. The two outer partitions also make up a software RAID 0, which is the meta-data device.

Ext2-1d Ext2 on one disk with only one partition.

Ext2-2d Ext2 on two disks, each one with only one partition. The two partitions make up a software RAID 0.

Ext3-1d Ext3 on one disk with only one partition.

Ext3-2d Ext3 on two disks, each one with only one partition. The two partitions make up a software RAID 0.

Contrary to that one can expect, DualFS, Ext2 and Ext3 have different behaviors for one disk, and for two disks. RAID 0 configurations [17] will help us to determine the strengths and weaknesses of each file system in a better way.

For DualFS, the meta-data device is always on the outer partition, since this partition is faster than the inner one. In addition, the cleaner is enabled in all tests, although it is hardly required because the number of clean segments is almost always above the threshold (10 segments).

All RAID 0 configurations have a chunk size of 128 KB. For DualFS, meta-data device is 10% of the total disk space. The logical block size is always 4 KB.

Separate Disks for Data and Meta-Data

Since DualFS puts data and meta-data on different devices, it is possible to put data on one disk, and meta-data on another disk. In order to calculate the improvement achieved by this separation, we have run the microbenchmarks for a new DualFS configuration. This configuration puts data on an entire disk, and meta-data on a partition of another disk, similar to the data one. The meta-data partition is 10% of the entire disk.

This configuration will also allow us to estimate how many times the meta-data device can be slower than the data device to achieve the same performance Ext2 obtains with a device similar to the data one. In this way, we will know if a disk is suitable to be used as the meta-data device.

Since some journaling file systems allow us to put the journal on a separate disk, it would be interesting to test this configuration for the journaling file system used in our benchmarks. Unfortunately, an Ext3 version with this feature only exists in Linux 2.4.

Table 1: System Under Test

	Linux Platform
Processor	Two 450 Mhz Pentium III
Memory	256 MB, PC100 SDRAM
Disk	Two 4 GB IDE 5,400 RPM Seagate ST-34310A. Two 4GB SCSI 10,000 RPM FUJITSU MAC3045SC. SCSI disk 1: Operating system, swap and trace log. SCSI disk 2: trace log. IDE disks: test disks
OS	Linux 2.2.19

4.4 Cleaner Evaluation

One of the main drawbacks of a log-structured file system is the cleaner. Since our meta-data device is implemented as an LFS, we must evaluate the impact of the cleaner on the performance. Most of the benchmarks described above are not suitable for evaluating the DualFS cleaner, because they do not produce enough half-full dirty segments. Hence, we need a new test.

In the experiment we have designed for evaluating the cleaner, a directory tree is copied and then, 87.5% (7/8) of its regular files are deleted. This process is repeated 20 times. This experiment is carried out for DualFS-1d under two configurations: without cleaner, and cleaning a segment every five seconds.

Cleaning a segment every five seconds is very intrusive, but it gives us a conservative estimation of the impact of the cleaner on performance. In a tuned prototype, the cleaner can be run when the storage system is idle. Hence, the cleaner impact will be lesser. This test is denoted *write-del*.

4.5 System Under Test

All tests are done in the same machine. The configuration is shown in Table 1.

In order to trace disk activity, we have instrumented the operating system (Linux 2.2.19) to record when a request starts and finishes. The messages generated by our trace system are logged in an SCSI disk which is not used for evaluation purposes.

Messages are printed using the kernel function *printk*. This function writes messages in a circular buffer in main memory, so the delay inserted by our trace system is small ($< 1\%$), especially if compared to the time needed to perform a disk access. Later, these messages are read through the `/proc/kmsg` interface, and then written to the SCSI disk in big chunks. In order to avoid the loss of messages (last messages can overwrite the older ones), we have increased the circular buffer from 16 KB to 1 MB, and we have given maximum priority to all processes involved in the trace system.

5. EXPERIMENTAL RESULTS

We are interested in the total time taken for all disk I/O operations. The total I/O time gives us an idea of how much 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.

We have represented the confidence intervals for the mean as error bars, for a 95% confidence level. For comparison purposes between figures, the absolute Ext2 I/O time has been written inside each Ext2 bar. The numbers inside the other bars are the I/O times, normalized with respect to the Ext2 I/O time.

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

5.1 Microbenchmarks

Microbenchmarks results can be seen in Figure 3. The six benchmarks have been run for 1 and 2 disks, and for 1 and 4 processes.

Data Benchmark Results (-dm tests)*

Figure 3 shows that DualFS is always better than Ext2 and Ext3 in write tests. In these tests, the average request size in DualFS is greater than the average request size in Ext2. This is because DualFS writes all meta-data blocks to the log in big chunks, whereas Ext2 writes meta-data blocks in small chunks, since its meta-data blocks are spread across the disk. If the average request size is greater, the number of requests is smaller. Hence, the total I/O time is smaller in DualFS than in Ext2, even though the average access time in the former is a little greater than in the latter.

Ext3 has the same problem Ext2 has with meta-data writes. In addition, Ext3 has to write meta-data blocks in its log too, and these writes can cause long seeks (note that the Ext3 log, as the log of many other journaling file systems, is in a fixed disk location). Therefore, the total I/O time of Ext3 is greater than the Ext2 one, and much greater than that of DualFS.

In read tests, Ext2 and Ext3 have similar performance, as expected. On the other hand, DualFS performance is significantly worse than Ext2 and Ext3 performance only in one test. This is surprising because DualFS has data and meta-data blocks in different partitions, and this would have to cause long seeks. Hence, DualFS performance would have to be bad.

The explanation is that the *read-data-meta* test has a lot of meta-data writes because the access time field in the i-node of every file which is read must be updated. These meta-data writes are very fast in DualFS. The same does not occur for Ext2 and Ext3. For example, in Ext2, all the write requests take 1.5 seconds (7.5% of the total I/O time) for 1 process and 1 disk. In Ext3, this time is 1.3 seconds (6.1% of the total I/O time). And it is less than 1% in DualFS.

Increasing the number of processes makes DualFS to perform significantly worse than Ext2 and Ext3. When the number of processes goes from 1 to 4, the average read access time increases for the three file systems. However, this increase is a little bigger in DualFS than in Ext2 and Ext3 due to the separation between data and meta-data.

The case is different when we add a second disk, since DualFS beats Ext2 and Ext3 in spite of the separation between data and meta-data. Adding a second disk damages Ext2 performance, since related blocks that are together when we have one disk can be separate when we have two disks. Think about the data and meta-data blocks of a file. If we want to read the block 'n' of a file, we must read first some

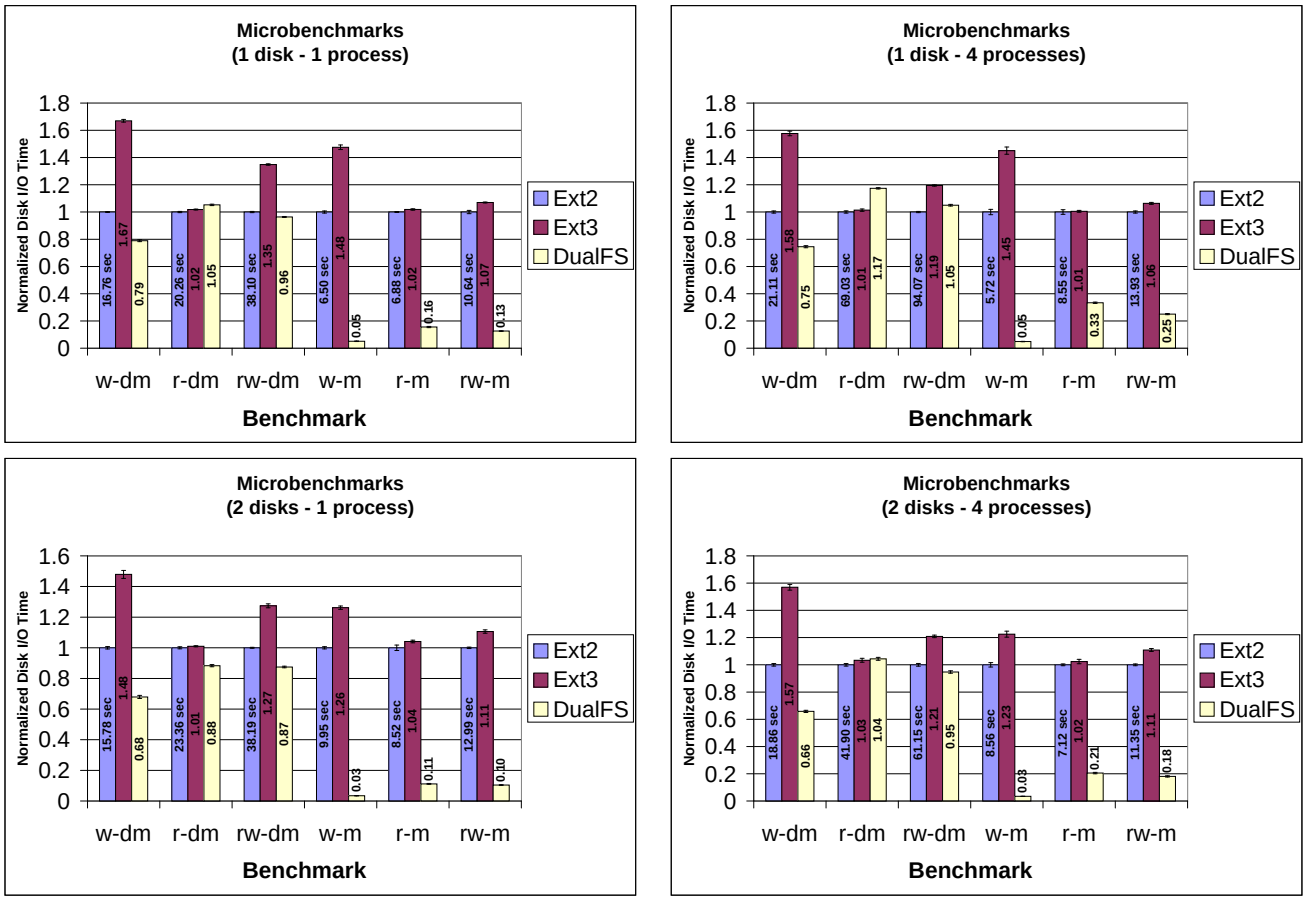

Figure 3: Microbenchmarks results.

meta-data blocks of the file, and then the data block. If we have one disk, we will have to do only one seek (or two seeks, but the second one will be very short). If we have two disks, the meta-data blocks can be on one disk, and the data block on the other disk, so we will have to do two seeks. Ext3 suffers the same problem. However, this is not a problem for DualFS because there are already two long seeks when we have one disk (due to the separation between data and meta-data blocks), and the second disk does not increase the number of long seeks.

Finally, we can see that the results of every read/write test are a mixture of the results of its respective read and write tests.

Meta-data Benchmark Results (-m tests)*

Figure 3 shows that DualFS clearly beats both Ext2 and Ext3 in all meta-data only tests.

Since the DualFS meta-data device is a sequential log, meta-data writes in DualFS are done in big chunks, and at a rate close to the maximum disk bandwidth. In Ext2 and Ext3, meta-data blocks are spread across the storage device. Hence, meta-data writes are done in small requests which have a long access time. Ext3 also has to write meta-data blocks in the log, so its total I/O time is greater than the Ext2 one.

In the read test, Ext2 and Ext3 have similar performance, as expected. However, DualFS performance is incredibly good. There are three reasons for that performance: the high meta-data locality achieved by DualFS in the log, where related meta-data blocks are written together, the disk built-in cache, which prefetches up to 512 KB of meta-data, and the small size of the meta-data device, what makes seeks shorter. There is also meta-data locality in Ext2 and Ext3, since related meta-data blocks are in the same group. However, meta-data blocks are spread across the group, whose size (128 MB) prevents the disk built-in cache from catching that locality.

Increasing the number of processes harms DualFS performance, although it is still much better than that of Ext2 and Ext3. When the number of processes is 4, there is less meta-data locality in the read stream. Therefore, the disk built-in cache is less efficient. This is a problem for DualFS, which loses much of the locality achieved for one process, but not for Ext2 and Ext3, where the built-in cache is already little efficient for one process, as we have seen above.

The effect of adding a second disk on the read test depends on the number of processes. When the number of processes is one, only DualFS is able to take advantage of both disks, since the built-in cache is greater (1 MB). However, none of the three file systems can take advantage of the parallelism offered by both disks. When there are four processes, the three file systems can exploit the parallelism offered by the two disks, and DualFS again benefits from the large built-in cache.

Finally, in the read/write tests, the results obtained are a mixture of the results of the respective read tests and write tests.

CPU Utilization

Table 2 shows the CPU utilization of every file system in the *read-write-data-meta* test, for one disk and one process. Each CPU time value is the sum of the user time and system time. Confidence intervals are given as percentage of the CPU time.

Although the CPU utilization of DualFS is bigger than the one of Ext2 and Ext3, this time is small when compared with the total I/O time. We must also take in account that DualFS computes two checksums when it writes a partial segment. Neither Ext2 nor Ext3 compute checksums. Finally, it is sure that our prototype is not as tuned as the Ext2 and Ext3 implementations.

5.2 Macrobenchmarks

Results of macrobenchmarks are shown in Figure 4, for one and two disks.

DualFS clearly beats Ext2 and Ext3 in the kernel compilation, Specweb99, and PostMark tests. This is because there are a lot of write operations in these tests, and DualFS improves write operations greatly, especially in workloads where Ext2 and Ext3 have data and meta-data blocks spread across the disk.

In our video compression test, however, Ext2 wins because this test is a very good one for Ext2. This test only uses 2 directories: one for files to be read, and another for files to be written. Hence, data and meta-data are not spread across the device, and Ext2 is not obligated to do long seeks to write, and read, data and meta-data. DualFS has to do long seeks due to the separation between data and meta-data blocks, but its performance is still much better than Ext3, because Ext3 also has to do long seeks between the journal and the regular file system.

As in the microbenchmark case, DualFS performance improves more than that of Ext2 and Ext3 when the number of disks goes from 1 to 2 (except for the PostMark and video compression tests). Since DualFS enforces a stricter order among write requests, these are done in large chunks and at more specific moments. This allows DualFS to make better use of the parallelism offered by the two disks which make up the RAID 0 devices. In the PostMark test, however, Ext2 does almost five times as many write requests as DualFS. Hence, the parallelism provided by both disks has a greater effect on Ext2 than on DualFS. For Ext3 is the same.

5.3 Separate Disks for Data and Meta-Data

Results of this configuration are shown in Figure 5. As we can see, there are three bars for every benchmark. The first bar is the Ext2 I/O time. The second one is the I/O time taken by the meta-data accesses in DualFS. And the third one is the I/O time taken by the data accesses in DualFS.

Table 2: CPU utilization

	Ext2	Ext3	DualFS
CPU Time (secs)	3.82	4.80	6.54
Confidence Interval	4.45%	1.92%	1.22%

Note that the DualFS I/O time is less than the addition of the data I/O time and the meta-data I/O time, because of the overlap between data and meta-data accesses.

As expected, DualFS beats Ext2 in all cases. We can also see that the behavior when the number of processes goes from 1 to 4 is the same as the one seen in 5.1.

Taking into account that we want DualFS to beat Ext2 in all configurations, we are going to estimate when the meta-data device would be too old with respect to the data one to not meet that condition. This estimation is based on information in [7, 20]:

- capacity: if disk capacity doubled every 3 years, and the meta-data device size was 10% of the data device size, the meta-data device would be too small after 10 years.
- seek time: if seek time decreased 5% every year, the meta-data device would be too slow after more than 10 years.
- rotational speed: if rotational speed doubled every 6 years, the meta-data device would be too slow after 9.5 years.

Since disk throughput mainly depends both on seek time and latency time [15], it is quite safe to say that DualFS will beat Ext2 if it uses disks which are up to 9 or 10 years old as meta-data device. Even older disks could be used if we do not expect to get a gain in all workloads but just in most of them. Although this usage of old disks might not be very important for high-performance super-computing centers, it might be very useful in low-cost systems and in home computers where the disk from the previous system can be used to improve performance instead of throwing it away.

5.4 Cleaner Evaluation

The results of the *write-del* test are shown in Table 3. As we can see, the cleaner, even when it is very intrusive, has a very small impact on the DualFS performance.

Although DualFS is much better than Ext2 in write tests, in this test it is only slightly better. The main reason is that meta-data writes are more frequent in DualFS than in Ext2. In DualFS (as in Ext3), a dirty meta-data block can be up to 5 seconds in main memory, whereas it can be up to 10 seconds in main memory if the file system used is Ext2. Since DualFS flushes data updates to disk before any transactions commit, the data blocks of many files which will be deleted after, will be flushed to disk. Many of these data blocks will be assigned to new files in the next step, and written to disk again. In Ext2, however, these data blocks will be freed, and assigned to new files, before being written to disk. Hence, these blocks will be written once, instead of twice. This causes the amount of blocks written to disk to be smaller in Ext2 than in DualFS, although write requests are greater in DualFS.

Based on these results, we can suppose that the cleaner will not be a great problem for DualFS. Since meta-data is a small part of a file system (10%, or less), if the meta-data device is large enough there will be always a lot of clean segments (or a lot of dirty segments with few live bytes), and the cleaning will take little time.

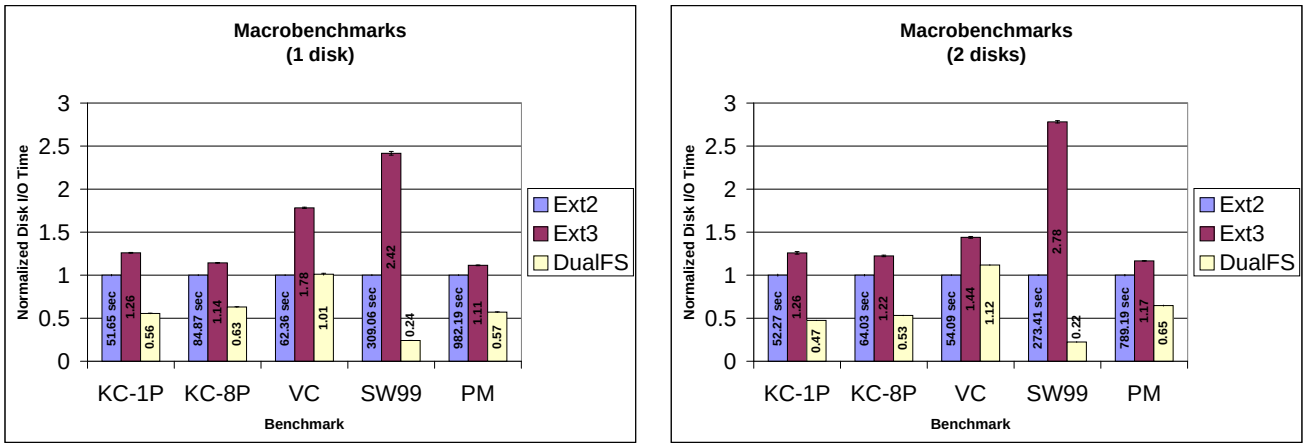

Figure 4: Macrobenchmarks results.

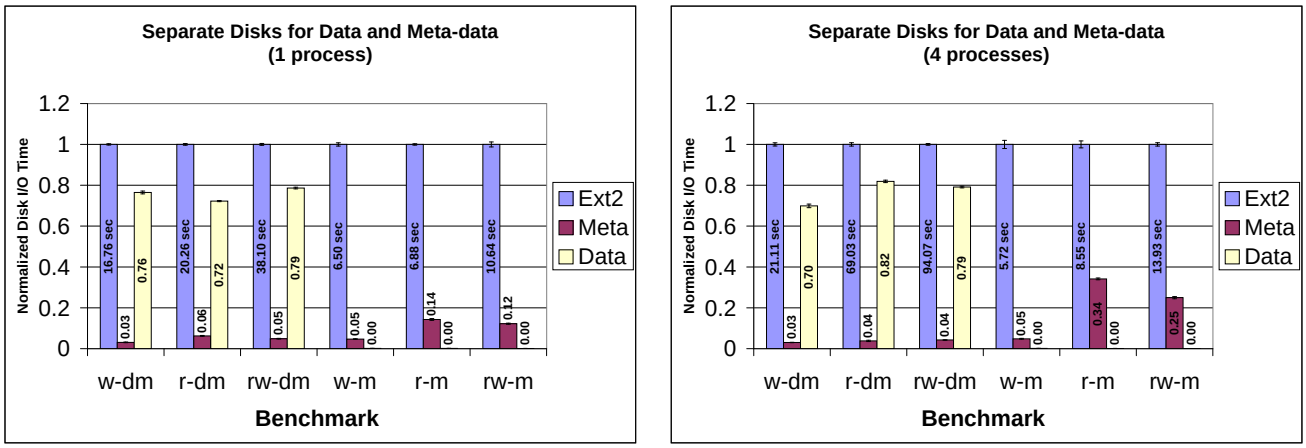

Figure 5: Using one disk for data and another one for meta-data.

6. CONCLUSIONS

In this paper we have introduced DualFS, a new journaling file system that manages, in very different ways, data and meta-data. The new file system separates completely data and meta-data, and places them on different devices. While data is organized much as it is by a traditional Unix file system, meta-data is organized like a log-structured file system. This log allows DualFS a quick consistency recovery after a system crash.

The main difference between DualFS and other journaling file systems is the fact that DualFS has only one copy of every meta-data block, whereas other journaling file systems has two copies. In avoiding an expensive extra copy of meta-data blocks, DualFS can achieve a good performance if compared with other journaling file systems.

Table 3: Results of the *write-del* test

File System	Total I/O Time (secs)	Confidence Interval
Ext2	44.90	0.42%
Ext3	64.04	2.59%
DualFS – Cleaner	42.16	1.61%
DualFS + Cleaner	44.41	2.01%

We have compared DualFS against Ext2, a FFS-like file system, and Ext3, a journaling file system derived from Ext2. Our experimental results show that DualFS greatly reduces the total I/O time taken by the file system in most cases (up to 97%), whereas it increases the total I/O time in a few and limited cases (up to 17%).

Since meta-data is organized as a log-structured file system, a cleaner is needed. We have evaluated the impact of the cleaner on DualFS performance, and we have found that it is very small (less than 6%).

Finally, we have also evaluated the possibility of using two identical disks: one for data and one for meta-data. In this configuration, DualFS beats Ext2 in all cases, and its performance improvement on Ext2 ranges from 15% to 95%. If an old disk is used for meta-data, it can be up to 9 or 10 years old with respect to the data disk before being too slow for DualFS beating Ext2. Unfortunately, we have not been able to do this test for Ext3.

The case where DualFS is not always better than Ext2 and Ext3, is the read of data and meta-data. Nevertheless, we think that the high meta-data locality achieved for DualFS can be exploited to improve DualFS performance. We are working on this issue.

6.1 Availability

For more information about DualFS, please visit the Web site at `http://www.ditec.um.es/~piernas/dualfs`.

7. REFERENCES

[1] D. C. Anderson, J. S. Chase, and A. M. Vahdat. Interposed request routing for scalable network storage. *In Proc. of the Fourth USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 259–272, Oct. 2000.

[2] R. Card, T. Ts'o, and S. Tweedie. Design and implementation of the second extended filesystem. *In Proc. of the First Dutch International Symposium on Linux*, December 1994.

[3] P. M. Chen, W. T. Ng, S. Chandra, C. Aycock, G. Rajamani, and D. Lowell. The rio file cache: Surviving operating system crashes. *In Proc. of Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 74–83, Oct. 1996.

[4] S. Chutani, O. T. Anderson, M. L. Kazar, B. W. Leverett, W. A. Mason, and R. Sidebotham. The Episode file system. *In Proc. of the Winter 1992 USENIX Conference: San Francisco, California, USA*, pages 43–60, January 1992.

[5] G. R. Ganger and M. F. Kaashoek. Embedded inodes and explicit grouping: Exploiting disk bandwidth for small files. *In Proc. of the USENIX Annual Technical Conference, Anaheim, California USA*, pages 1–17, January 1997.

[6] G. R. Ganger and Y. N. Patt. Metadata update performance in file systems. *In Proc. of the First USENIX Symposium on Operating Systems Design and Implementation (OSDI): Monterey, California, USA*, pages 49–60, Nov. 1994.

[7] E. G. Grochowski and R. F. Hoyt. Future trends in hard disk drives. *IEEE Transactions on Magnetics*, 32(3), May 1996.

[8] Y. Hu, Q. Yang, and T. Nightingale. Rapid-cache – a reliable and inexpensive write cache for disk I/O systems. *In Proc. of the Fifth Int. Symp. on High-Performance Computer Architecture*, pages 204–213, January 1999.

[9] JFS for Linux. http://oss.software.ibm.com/jfs, 2002.

[10] J. Katcher. PostMark: A new file system benchmark. *Technical Report TR3022. Network Appliance Inc.*, october 1997.

[11] T. Kowalski. Fsck: The UNIX system check program. *ACM Transactions on Computer Systems*, 2(3):181–197, Aug. 1978.

[12] M. McKusick, M. Joy, S. Leffler, and R. Fabry. A fast file system for UNIX. *ACM Transactions on Computer Systems*, 2(3):181–197, Aug. 1984.

[13] M. K. McKusick and G. R. Ganger. Soft updates: A technique for eliminating most synchronous writes in the fast filesystem. *In Proc. of the 1999 USENIX Annual Technical Conference: Monterey, California, USA*, June 1999.

[14] K. Muller and J. Pasquale. A high performance multi-structured file system design. *In Proc. of 13th ACM Symposium on Operating Systems Principles*, pages 56–67, Oct. 1991.

[15] S. W. Ng. Advances in disk technology: Performance issues. *IEEE Computer*, 31(5):75–81, May 1998.

[16] T. Olivares, F. J. Quiles, P. Cuenca, L. Orozco-Barbosa, and I. Ahmad. Study of data distribution techniques for the implementation of an mpeg-2 video encoder. *In Proc. of the Eleventh IASTED International Conference. MIT, Cambridge, Massachusetts (USA)*, pages 537–542, November 1999.

[17] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks (RAID). *In Proc. of the International Conference on Management of Data, ACM Press*, pages 109–116, 1988.

[18] J. K. Peacock, A. Kamaraju, and S. Agrawal. Fast consistency checking for the Solaris file system. *In Proc. of the USENIX Annual Technical Conference: New Orleans, Louisiana, USA*, pages 77–89, June 1998.

[19] M. Rosenblum and J. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, 10(1):26–52, Feb. 1992.

[20] Seagate Technology LLC. http://www.seagate.com, 2001.

[21] M. Seltzer, K. Bostic, M. K. McKusick, and C. Staelin. An implementation of a log-structured file system for UNIX. *In Proc. of the Winter 1993 USENIX Conference: San Diego, California, USA*, pages 307–326, January 1993.

[22] M. Seltzer, K. A. Smith, H. Balakrishnan, J. Chang, S. McMains, and V. Padmanabhan. File system logging versus clustering: A performance comparison. *In Proc. of the 1995 USENIX Technical Conference: New Orleans, Louisiana, USA*, pages 249–264, Jan. 1995.

[23] M. I. Seltzer, G. R. Ganger, M. K. McKusick, K. A. Smith, C. A. N. Soules, and C. A. Stein. Journaling versus soft updates: Asynchronous meta-data protection in file systems. *In Proc. of the 2000 USENIX Annual Technical Conference: San Diego, California, USA*, June 2000.

[24] Linux XFS. http://linux-xfs.sgi.com/projects/xfs, 2002.

[25] SPECweb99 Benchmark. http://www.spec.org, 1999.

[26] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck. Scalability in the XFS file system. *In Proc. of the USENIX 1996 Annual Technical Conference: San Diego, California, USA*, January 1996.

[27] S. Tweedie. Journaling the Linux ext2fs filesystem. *In LinuxExpo'98*, 1998.

[28] U. Vahalia, C. G. Gray, and D. Ting. Metadata logging in an NFS server. *In Proc. of the 1995 USENIX Technical Conference: New Orleans, Louisiana, USA*, pages 265–276, Jan. 1995.

[29] Veritas Software. The VERITAS File System (VxFS). http://www.veritas.com/products, 1995.

[30] R. Y. Wang, T. E. Anderson, and D. A. Patterson. Virtual log based file systems for a programmable disk. *In Proc. of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI'99)*, pages 29–43, Feb. 1999.