Regular Paper

dajFS: A New File System with Per-directory Adaptive Journaling

Wataru Aoyama[1,a)] Hideya Iwasaki[1,b)]

Received: October 26, 2018, Accepted: January 15, 2019

Abstract: A journaling file system is a file system that records information about pending updates to the file system before committing the updates. This mechanism raises the reliability of the system because it enables any inconsistencies to be repaired with minimal loss of data. Since there is a tradeoff between the overhead and reliability, ext3, a journaling file system commonly used by the Linux kernel, offers three journaling modes: speed-prioritized mode, reliability-prioritized mode, and intermediate mode. Unfortunately, in ext3, the journaling mode has to be set individually for each file system. Thus, the granularity of the journaling mode setting is very coarse. In addition, the journaling mode must be determined at the time of mounting the file system and cannot be changed without unmounting it. To resolve this problem, this paper proposes a new journaling file system named dajFS (per-directory adaptive journaling file system) that is able to set an appropriate journaling mode for each directory and to switch the journaling mode of a directory to another on the fly without unmounting the file system. Essentially, the journaling mode that is specified for a directory applies to all files that reside directly under that directory. By using dajFS, the user can determine and set a journaling mode for each directory on the basis of the importance of files under that directory. As a result, the user can enjoy moderate granularity with the journaling mode setting.

Keywords: journaling file system, consistency, journaling mode, ext3, Linux kernel

1. Introduction

File systems are the part of an operating system that manage data on storage devices. When an operating system is abnormally aborted due to, for example, power failure or system crash during a file operation, a file system might fall into an inconsistent state. Basic techniques to cope with this problem include file recovery based on file-system scan such as fsck [1], journaling [2], soft-update [3], [4], and copy on write [5]. Among these, journaling is used in various file systems such as the third extended (ext3) file system, JFS [*1], XFS [6], ReiserFS [*2], and NTFS [7].

A *journaling file system* is a file system that records information about pending updates to the file system before committing the updates. By this mechanism, it is possible to repair any inconsistencies with minimal loss of data.

Typically, a journaling file system logs metadata (i.e., information about actual data)/actual data into a special log file called a *journal*. After confirming that they are recorded in the journal, it writes them into the permanent storage on a disk and removes the logs from the journal. Note that we use "permanent storage" to represent the disk area in which metadata and actual data are permanently stored; this does not include the disk area for a log file. Even if the operating system crashes, it is possible to know how far the update process has proceeded by investigating the content of the journal, and thus to resolve any inconsistencies. In this paper, we use the term "log file" to represent a journal so as to avoid confusion because the term "journal" is also used as a journaling mode name.

The degree to which the inconsistencies can be resolved depends on how the data are recorded into a log file, i.e., *journaling mode*. In ext3, a journaling file system commonly used by the Linux kernel, three journaling modes are available: *writeback*, *ordered*, and *journal*. In this paper, we use slanted fonts to represent the name of a journaling mode. These journaling modes are different in their risk levels: *writeback* has the highest risk, *ordered* has the medium risk, and *journal* has the lowest. The risk level depends on which data are recorded and in what order they are recorded into a log file. Lower risk level means higher reliability, but also larger overhead.

Ideally, an adequate journaling mode with *moderate granularity* depending on the target risk level desired by the user is applied. For example, we want to apply *journal* for files that have important data, and to use *writeback* for files that are not so important. Unfortunately, in ext3, the granularity is "per file system", which means that journaling mode must be set individually for each file system. This granularity is too coarse, because *all* files in the same file system are applied using the same journaling mode. It is impossible to set an appropriate journaling mode in a finer way. For example, it is impossible to apply the *journal* mode to one part of a file system and to apply the *ordered* mode to another part at the same time. In addition, it is impossible to switch the journaling mode without unmounting the file system first.

A few studies have removed this restriction of ext3. For example, File-adaptive Journaling [8] is a file system that supports

[1] The University of Electro-Communications, Chofu, Tokyo 182–8585, Japan
[a)] aoyama@ipl.cs.uec.ac.jp
[b)] iwasaki@cs.uec.ac.jp

[*1] http://jfs.sourceforge.net/
[*2] https://reiser4.wiki.kernel.org/index.php/Main_Page

per-file journaling mode by enabling the user to select an adequate journaling mode for each file. Unfortunately, choosing and setting a journaling mode for *every* file is a very time-consuming task for the user. Thus, the per-file granularity is too fine.

To resolve this problem, we propose a new journaling file system named dajFS (per-directory adaptive journaling file system) that sets an appropriate journaling mode for each *directory* and can switch the journaling mode of a directory to another on the fly without having to unmount the file system. By using dajFS, the user can easily set journaling modes with moderate granularity.

2. The ext3 Journaling File System

The ext3 is a journaling file system that was developed on the basis of ext2. In this section, we describe the internals of ext3, as we implemented dajFS as an extension of ext3.

2.1 Layout of ext3

The layout of ext3 is shown in **Fig. 1**, where data structures unrelated to this paper have been omitted. The ext3 manages a file system by dividing it into *block groups* of the same size [9]. The contents of each block group are as follows. I-node bitmap (IB) and data bitmap (DB) are bitmaps for managing i-node blocks and data blocks, respectively. I-node blocks (INODE) are blocks within which the i-nodes of files are stored. Data blocks (DATA) are blocks within which actual file data are stored. Among them, IB, DB, and INODE are classified as *metadata*. In this paper, *M* and *D* represent metadata and actual data, respectively. The gray part of the figure represents the area for a log file, the structure of which is described in Section 2.2.

To reduce the number of I/O operations, the Linux kernel caches data in a disk on the memory, which is called *disk cache*. We use the terms *i-node bitmap cache*, *data bitmap cache*, *i-node cache*, and *data cache* for the disk caches of IB, DB, INODE, and DATA, respectively.

When a user process issues an asynchronous write operation to a file, the kernel writes data to the corresponding disk caches and informs the user process of the completion of the write operation. As a result, every disk cache that includes new data is marked as "dirty", which means that the content of the disk cache has been changed and has to be flushed to the disk. Precisely speaking, there are two kinds of "dirty" disk cache. One is "JBDDirty", which is managed by the Journaling Block Device presented in Section 2.2, and the other is "normal dirty", which is managed outside the journaling mechanism. This paper does not distinguish these two and simply uses "dirty".

2.2 Structure of Log File

The ext3 uses a built-in kernel module for journaling called Journaling Block Device (JBD). JBD uses the following terms: *commit*, which is an operation for writing a data to a log file, *checkpoint*, which is an operation for writing a committed data in a log file onto a disk, and *recovery*, which is an operation for restoring the consistency of a file system in an inconsistent state.

A *transaction* is a group of file operations that should be performed as a whole as if it were an atomic operation. A transaction is managed mainly by two dually linked lists in the kernel: `t_buffers` and `t_sync_datalist`. The former holds every data that is to be written into a log file and the latter holds every data that is to be written directly into the permanent storage bypassing a log file. All data in `t_buffers` and `t_sync_datalist` are written to a disk by the kernel thread kjournald. By treating every data in `t_sync_datalist` as a part of a transaction, it is possible to enforce the writing order of data in both linked lists.

The structure of a log file in ext3 is presented in **Fig. 2**, which also omits the data structures unrelated to this paper. A log file consists of a journal superblock (JS) and a sequence of transactions that are used as a ring buffer. A journal superblock holds information that is necessary for managing a log file, such as the starting point of a sequence of transactions. In a transaction, change histories, i.e., dirty disk caches (M' and D'), are sandwiched between a descriptor block (JD) and a commit block (JC). The "prime" in M' and D' means "dirty": M' is a dirty metadata and D' is a dirty actual data. Each transaction is given a unique identifier (transaction ID). ext3 executes transactions in the order of this ID.

A transaction is one of three states: *running*, *committing*, and *checkpointing*. These states are shown in **Fig. 3**. A running transaction can add a new change history, i.e., a dirty disk cache, to `t_buffers`/`t_sync_datalist`. If a synchronization instruction is performed or timeout occurs, the running transaction transits to the committing state. In the committing state, a transaction is not allowed to add a new change history, and change histories in the transaction are committed one by one. In this committing process, every disk cache in `t_buffers` is written in a log file and every disk cache in `t_sync_datalist` is written in the per-

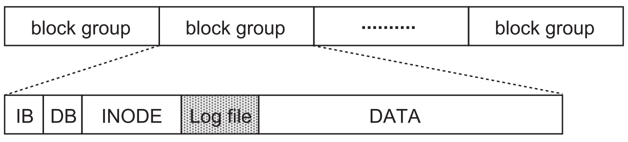


Fig. 1 Layout of ext3.

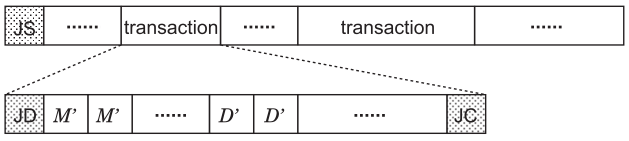
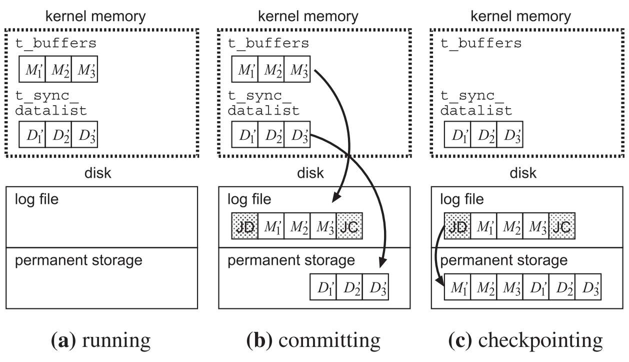


Fig. 2 Structure of log file in ext3.



(**a**) running (**b**) committing (**c**) checkpointing

Fig. 3 States of a transaction.

manent storage. When the commitment for all change histories is completed, the state transits to the checkpointing state. In this state, every data in a log file is written into the permanent storage. Once a transaction reaches the checkpointing state, it is possible to recover the file system even if the operating system abnormally crashes.

2.3 Journaling Modes

As discussed in the Introduction, there are three journaling modes in ext3: *writeback*, *ordered*, and *journal*. The operation flows for these modes are shown in **Fig. 4**.

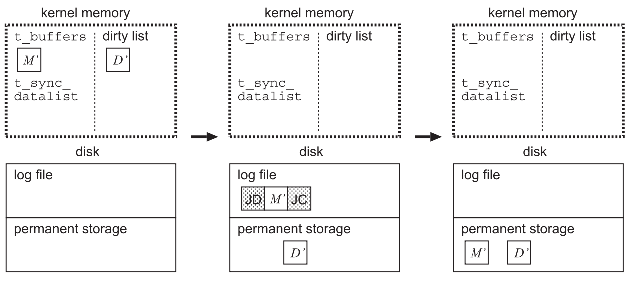
In the *writeback* mode (Fig. 4 (a)), ext3 writes disk caches for all kinds of metadata (i.e., i-node bitmap caches, data bitmap caches, and i-node caches) into a log file. In contrast, ext3 does not write any data caches into a log file but rather writes them into the permanent storage directly. This mode does not prescribe their writing order into a log file/permanent storage. For this purpose, ext3 places dirty disk caches for all kinds of metadata on `t_buffers` and dirty data caches for actual file data on a data structure called a *dirty list* that is outside the journaling mechanism. Since the writing order is not determined, a metadata on the permanent storage might come to refer to invalid actual data, but the consistency of metadata cannot be compromised.

The *ordered* mode (Fig. 4 (b)) also writes disk caches for all kinds of metadata into a log file and data caches for actual file data directly into the permanent storage. The difference between this mode and *writeback* is that *ordered* enforces the writing order: actual file data first and then metadata. To do so, ext3 places dirty disk caches for all kinds of metadata on `t_buffers` and dirty data caches on `t_sync_datalist`. Due to the enforcement of writing order, a metadata on the permanent storage is guaranteed to refer to a valid actual data.

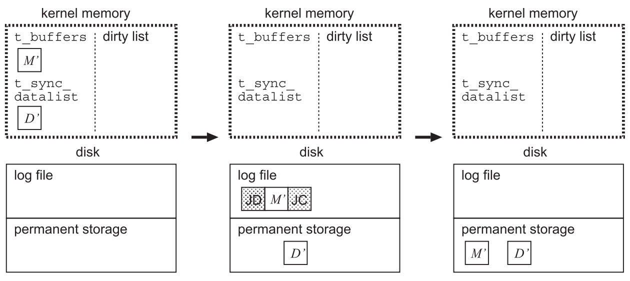
The *journal* (Fig. 4 (c)) writes all data caches (for both metadata and actual data) into a log file by placing them on `t_buffers`. Thus, the consistency of metadata and actual data cannot be compromised.

There is a tradeoff between the overhead and reliability. The *writeback* mode has the lowest overhead among the three but its reliability is also the lowest. In contrast, *journal* mode has the highest reliability at the expense of the speed: its overhead is the highest. The overhead and reliability of *ordered* lie between those of *writeback* and *journal*.

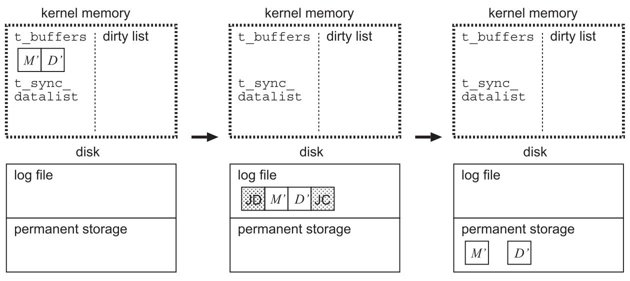
In ext3, journaling mode has to be set *per file system*; i.e., the journaling mode is determined at the time of mounting the file system and cannot be changed without unmounting it. Thus, the granularity of the journaling mode setting is very coarse. Some studies [10], [11] have focused on raising both the speed and reliability of the file system at the same time, but they cannot change the granularity of the journaling mode setting.



(a) *writeback* mode



(b) *ordered* mode



(c) *journal* mode

Fig. 4 Three journaling modes in ext3.

3. Design of dajFS

3.1 Overview and Design Decisions of dajFS

The dajFS is a file system that enables the user to set a journaling mode *per directory* in a file system and to switch the journaling mode of one directory to another on the fly without unmounting the file system. Basically, the journaling mode that is specified for a directory applies to *all* files that reside directly under that directory. We think that the design decision of per directory granularity of the journaling mode setting is quite adequate, for the following reason.

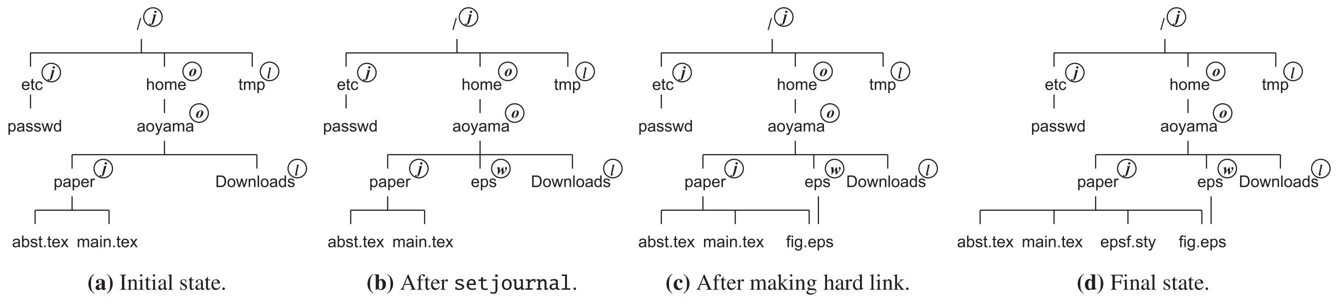
In Linux, standard directory names and contents placed under standard directories are determined on the basis of the Filesystem Hierarchy Standard [12]. For example, the `/etc` directory holds the system's configuration files and the `/tmp` directory holds temporary files. As you can see from these examples, files under the same directory tend as having common features, and consequently can be regarded as having almost the same importance. These observations led us to design the dajFS as a journaling file system whose granularity is per-directory. As discussed in the Introduction, the granularity of ext3 is the entire file system, which is too coarse. In contrast, per-file granularity [8] is too fine, even though it would be convenient for the user to be able to assign a journaling mode to each file. The problem is that specifying adequate journaling modes to all files would be a very time-consuming task for the user. Thus, per-directory granularity is a moderate and adequate design decision.

3.2 Journaling Modes in dajFS

The dajFS provides the same three journaling modes as those in ext3 along with one addition: the *lightweight* mode, which guarantees the minimum required ability for maintaining consistency by recording only i-node bitmap and data bitmap

Table 1 Journaling modes in dajFS.

journaling mode	target of log: i-node bitmap cache	data bitmap cache	i-node cache	data cache	order	overhead	consistency
lightweight	✓	✓				minimum	minimum
writeback	✓	✓	✓			small	small
ordered	✓	✓	✓		$D \rightarrow M$	medium	medium
journal	✓	✓	✓	✓		large	high



(a) Initial state. **(b)** After `setjournal`. **(c)** After making hard link. **(d)** Final state.

Fig. 5 Example of file manipulation in dajFS.

into a log file.

There are two situations in which the *lightweight* mode is required. The first is when there is a directory where even the consistency guaranteed by *writeback* mode is unnecessary. `/tmp` might be an instance of such directories. The second is when there are directories used by an application that has its own ways of guaranteeing consistency of the files related to the application. For example, SQLite assures consistencies of databases by treating the execution of an SQL statement as a transaction. For directories governed by such an application, any journaling mode in ext3 could be too much, as both the application and the file system make duplicated efforts to keep the consistencies. The *lightweight* mode can be used by such applications to reduce overheads caused by this duplication. If we use *lightweight* for such directories, we can entrust guaranteeing consistencies to the application's mechanism while minimizing the overhead of journaling by dajFS.

To sum up, the user can select one of four journaling modes — *lightweight*, *writeback*, *ordered*, or *journal* — for each directory depending on the importance of the files directly under that directory. These four journaling modes by dajFS are summarized in **Table 1**.

Similar to ext3, the user can use dajFS by first creating a file system on a block device by `mkfs` and then mounting the file system by `mount`. After that, the user can set/switch the journaling mode of a directory by using the interfaces provided by dajFS. By default, a new directory is given the same journaling mode on creation as that of its parent directory. The user can also change the journaling mode at any time while keeping the file system mounted.

3.3 Treatment of Links and File Movements

In Linux, it is possible to give multiple path names to a file by using hard links or symbolic links. This makes the relationship between files and path names one-to-many.

In dajFS, if a file is hard-linked from multiple directories, the file obeys the journaling mode of the directory that made the hard link most recently. For example, when a new hard link to a file is created, the journaling mode of the directory from which the link is created is applied to the file. This might change the journaling mode applied to the file. In contrast, when a symbolic link to a file is created, the journaling mode applied to the file is not affected.

When a file is moved from one directory to another, the file obeys the journaling mode of the latter directory.

3.4 Example of File Manipulation in dajFS

To illustrate how dajFS works, here we show a small and simple example. We assume a current user named `aoyama` whose home directory is `/home/aoyama`. **Figure 5** (a) shows a part of the directory structure of the file system rooted at / whose journaling mode is *journal*. The journaling mode of a directory is presented as the first letter of the journaling mode name in a circle at the top-right corner of the directory name.

Since the `/etc` directory contains important files such as the password file (`/etc/passwd`), its journaling mode is *journal*. In contrast, `/tmp` directory is set to *lightweight* because it generally contains less important temporary files. The importance of `/home` is between the two. Thus, we set its journaling mode and that of `aoyama` as *ordered*. There are two directories under the `aoyama` directory: `paper` and `Downloads`. The `paper` directory contains important source files of a paper, so it is assigned the *journal* mode. In contrast, `Downloads` contains files downloaded from the Internet, which can be redownloaded if necessary, so we select *lightweight* for `Downloads`. Initially, `paper` contained two files, `abst.tex` and `main.tex`, but `Downloads` currently has no file.

Suppose that the user is `aoyama` and initially the current working directory is `/home/aoyama`. Also suppose that the user executes the sequence of commands presented in **Fig. 6**. In this shell session, two commands provided by dajFS are used. First, "`lsjournal` *dir*" prints the journaling mode name of a specified directory to the standard output. If *dir* is omitted, the current working directory is used. Second, "`setjournal` *dir mode*" switches the journaling mode of the specified directory *dir* to *mode*. The result of switching is printed to the standard output.

```
$ pwd
/home/aoyama
$ lsjournal                Prints the journaling mode of /home/aoyama.
ordered                                  The journaling mode is ordered.
$ mkdir eps; cd eps                              Makes eps directory.
% lsjournal            Prints the journaling mode of /home/aoyama/eps.
ordered            The journaling mode is the same as that of the parent.
$ setjournal .  writeback                     Switches it to writeback.
.:  ordered -> writeback           Journaling mode switch completed.
Here draws a figure and stores the figure into fig.eps.
$ ls
fig.eps
$ cd ../paper
$ lsjournal          Prints the journaling mode of /home/aoyama/paper.
journal
$ ln ../eps/fig.eps fig.eps                         Makes a hard link.
Here downloads epsf.sty and stores it into /home/aoyama/Downloads.
$ mv ../Downloads/epsf.sty .              Moves the downloaded file.
```

Fig. 6 Shell session of the example.

In this shell session, the user makes the `eps` directory just under `/home/aoyama`. Since a new directory inherits the journaling mode of its parent directory, its journaling mode is *ordered*. Next, the user switches the journaling mode of `eps` to *writeback* by issuing the `setjournal` command. The directory structure at this moment is presented in Fig. 5 (b).

Next, suppose that the user invokes an application to draw a figure and stores a generated file (`fig.eps`) in the `eps` directory. The `fig.eps` file obeys the *writeback* journaling mode. Then, the user changes the current working directory to `paper`, whose journaling mode is *journal*, and makes a hard link to `fig.eps`. As a result, `fig.eps` is referred to from two directories. Since the most recent link was created from the `paper` directory, `fig.eps` comes to obey the journaling mode of `paper`, i.e., *journal*. The directory structure at this moment is presented in Fig. 5 (c).

Finally, the user downloads `epsf.sty` from the network into the `Downloads` directory. At first, the downloaded file is managed by *lightweight*, which is the journaling mode of the `Downloads` directory. However, after moving the file to the `paper` directory by the `mv` command, `epsf.sty` comes to obey the `paper` directory's *journal* mode. The final directory structure is presented in Fig. 5 (d).

4. Implementation of dajFS

We implemented dajFS by extending the ext3 and Journaling Block Device (JBD) in Linux Kernel 4.2.

4.1 Remembering Journaling Mode

To hold a specified journaling mode in a data structure for a directory, we added journaling mode information in the i-node block on the disk and i-node cache. For the i-node block, we utilized two unused bits in the i-node flag area named `i_flags`, which has a width of 32 bits.

To reduce the overhead caused by referring to the directory's journaling mode during file manipulation, dajFS automatically caches the referred journaling mode in the i-node for the file. There are three timings for making this cache. First, when a file and an i-node for the file are newly created, dajFS stores the journaling mode of the parent directory into the i-node. Second, when a file is moved from one directory to another, dajFS caches the journaling mode of the destination directory. Third, when the journaling mode of a directory is changed, dajFS caches the new journaling mode for the files under that directory.

4.2 Interfaces for Getting/Setting Journaling Mode

dajFS serves interfaces that use the `ioctl` system call for setting/getting the journaling mode of a directory to the user. The new commands (arguments) given to `ioctl` are `IOC_GETJOURNAL` and `IOC_SETJOURNAL`.

Processing `IOC_GETJOURNAL` is quite simple. After making sure the target of the command is really a directory, dajFS returns the journaling mode stored in the i-node flags, i.e., `i_flags`. For the case of `IOC_SETJOURNAL`, after making sure that the target is a directory, dajFS stores the new mode given as an argument of `ioctl` into the i-node flags of the directory. In the latter case, before storing the new mode, it is necessary to commit and checkpoint all the transactions managed by dajFS so as to preserve the writing order within a transaction. If committing and checkpointing were neglected, the writing order within a transaction for a file whose journaling mode is switched could be disturbed, as dirty disk caches are placed in either `t_buffers` or the dirty list depending on the journaling mode (see Fig. 4) and disk writings from these lists are not synchronized.

4.3 Implementing *writeback*/*ordered*/*journal* Modes

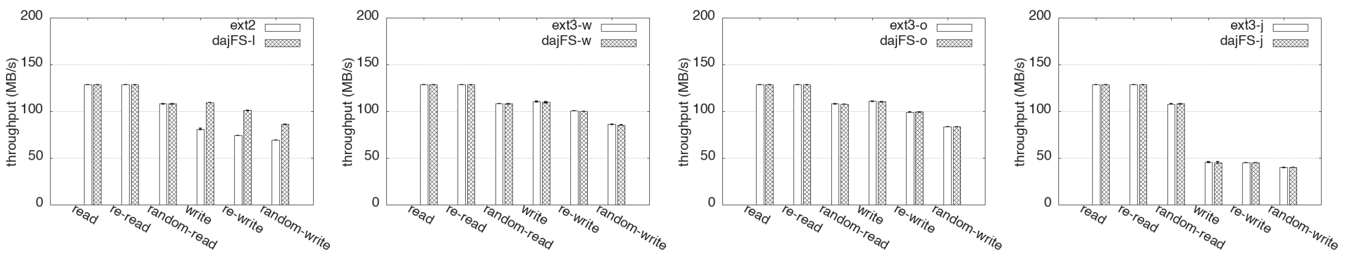
For *writeback*, *ordered*, and *journal* modes, we used ext3 implementations without any modifications. In ext3, functions in the kernel used for file operations never change once the file system is mounted. In dajFS, since journaling mode is not fixed, we implemented a mechanism for selecting appropriate functions used for file operations depending on the current journaling mode in the i-node cache of a target file.

4.4 Implementing *lightweight* Mode

A read operation in the *lightweight* mode is the same as those in the other three modes. It refers to the corresponding data cache for the target of the read operation from the i-node cache. If there exists no such data cache, or if the existing data cache is old, it reads the actual data from a disk.

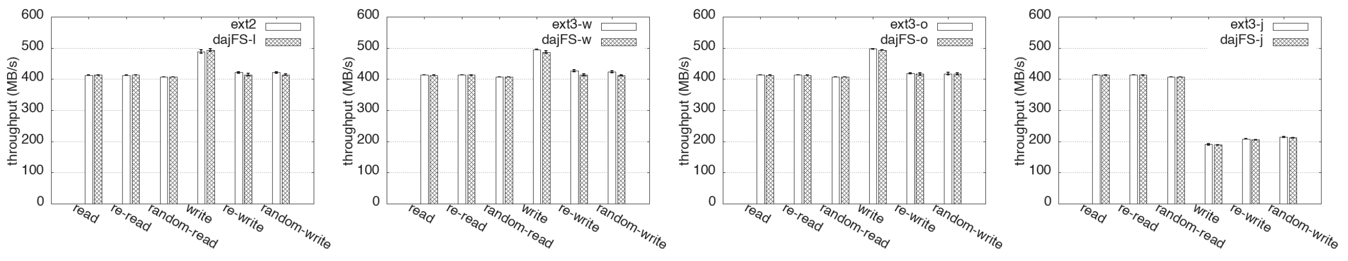
In a write operation in the *lightweight* mode, the targets of recording into a log file are i-node bitmap cache and data bitmap cache. Thus, dajFS places them in `t_buffers`. In addition, similar to the data caches in the *ordered* mode, dajFS places i-node caches in `t_sync_datalist`, whose contents are not written into a log file but are directly written into the permanent storage. This enables us to implement the *lightweight* mode on the basis of the code for the *writeback* mode.

It should be noted that, in the *lightweight* mode, there might be a case where a transaction has no dirty disk caches in `t_buffers`. Such a case could not happen in other journaling modes because if a write operation makes a data cache dirty, it definitely makes its corresponding i-node cache dirty at the same time. If we only used the kjournald kernel thread of ext3 without any modifications, a meaningless transaction that has only JD and JC might



(**a**) *lightweight* mode (**b**) *writeback* mode (**c**) *ordered* mode (**d**) *journal* mode

Fig. 7 Results of IOZone tests for HDD.



(**a**) *lightweight* mode (**b**) *writeback* mode (**c**) *ordered* mode (**d**) *journal* mode

Fig. 8 Performances of IOZone tests for SSD.

Table 2 System configuration used in the evaluation.

Hardware	CPU	Intel Core i5-4570 3.20 GHz 8M cache
	Memory	8 GB
	HDD	500 GB Hitachi Serial ATA/3.0 7200 rpm
	SSD	120 GB Intel SSD 520 Series
Software	Linux	Ubuntu16.04 x86_64
	kernel	Kernel 4.2

be produced in a log file. Thus, we modified kjournald so that it avoids producing such a meaningless transaction.

5. Evaluation

To determine the effectiveness of the proposed dajFS, we conducted experiments from the following two viewpoints.

- overhead of dajFS compared to ext3/ext2
- validity of per-directory journaling mode of dajFS

Table 2 presents the system configuration of the experiments. We prepared two environments: one using HDD and the other using SSD as an external storage, on which a file system (dajFS/ext3/ext2) was created. We executed every experiment five times and calculated their average values.

We used the following file systems.

- dajFS-l, dajFS-w, dajFS-o, and dajFS-j: dajFS where the journaling mode of the parent directory of target files for operation (e.g., write) was set to *lightweight*, *writeback*, *ordered*, or *journal*, respectively.
- ext3-w, ext3-o, and ext3-j: ext3 when journaling mode for a file system was set to *writeback*, *ordered*, or *journal*, respectively.
- ext2: ext2 as a journaling-less file system.

5.1 Overheads

To evaluate the overhead of dajFS, we used the IOZone file system benchmark tool [*3] and measured the I/O performances of read, re-read, random-read write, re-write, and random-write tests.

In the experiments, file size was set to 16 GB and the unit size of a single write was set to 8 MB. The obtained performance took the elapsed times by `close`, `fsync`, and `fflush` into account.

We compared dajFS-w with ext3-w, dajFS-o with ext3-o, and dajFS-j with ext3-j. dajFS-l was compared with ext2 because there is no corresponding journaling mode to *lightweight* in ext3.

Figures 7 and **8** present the results. As shown, dajFS had almost the same performances in both HDD and SSD environments as those by ext3 for all tests in *writeback* (Fig. 7 (b) and Fig. 8 (b)), *ordered* (Fig. 7 (c) and Fig. 8 (c)), and *journal* (Fig. 7 (d) and Fig. 8 (d)) modes. These results demonstrate that the introduction of per-directory journaling mode in dajFS imposed no extra overheads compared with ext3.

For *lightweight* mode (Fig. 7 (a) and Fig. 8 (a)), which has been newly introduced into dajFS, the performances of read, re-read, and random-read tests for both HDD and SSD and those of write, re-write, and random-write tests for SSD were almost the same as those by ext3. For the performances for write, re-write, and random-write tests for HDD, dajFS-l outperformed ext2 dramatically. This was because a specialized tree data structure called HTree was introduced in ext3, the base of dajFS, to promote efficiency of the file system.

5.2 Validity of Per-directory Granularity

To determine whether the per-directory granularity of journaling mode setting in dajFS is reasonable, we measured the performances of two benchmark programs: SQLite [*4] and MySQL [*5]. When using dajFS, we assigned appropriate journaling modes to various directories without losing the integrity of the systems.

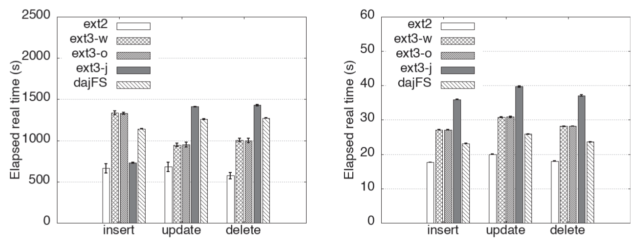
5.2.1 SQLite

SQLite is an open-source relational database management sys-

[*3] http://www.iozone.org

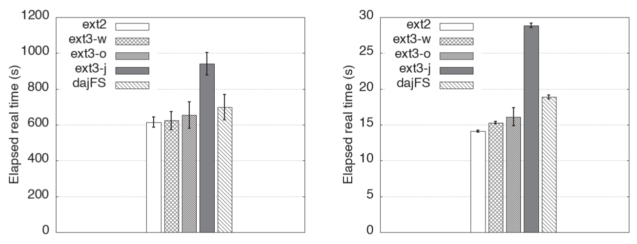
[*4] https://www.sqlite.org/

[*5] http://www.mysql.com/



(a) HDD **(b)** SSD

Fig. 9 Elapsed times of SQLite.



(a) HDD **(b)** SSD

Fig. 10 Elapsed times of MySQL.

tem that has its own internal mechanism for guaranteeing the database integrity. At the beginning of a transaction, SQLite creates a temporary file called *rollback journal file*, and uses it to roll back the database when abnormality occurs. SQLite writes every modification into the rollback journal file, and upon commitment, reflects the modification to the database. Thus, two duplicated mechanisms for maintaining consistency work at the same time: one served by a journaling file system (ext3/dajFS) and the other served by SQLite itself.

To avoid such duplication, we made small modifications to the source code of SQLite so as to place the database itself and the rollback journal file on different directories. When using dajFS, we set the journaling mode of the database directory as *journal* and that of the directory of the rollback journal file as *lightweight*.

In these experiments, for a table of integers and strings, we measured elapsed time for 10,000 insertions of random data, that for 10,000 random updates of stored data, and that for 10,000 random deletions of stored data.

Figure 9 presents the results for the HDD and SSD environments when the journal mode served by SQLite is set to its default [*6].

Elapsed times for random updates by dajFS were 11% shorter for HDD and 35% shorter for SSD than those of ext3-j. In addition, elapsed times for random deletions by dajFS were also 11% shorter for HDD and 36% shorter for SSD than those of ext3-j. In contrast, dajFS showed worse performance than ext3-j for random insertions to HDD. We presume this is because ext3-j treated actual data as the target of journaling, which accelerated sequential writes of blocks.

These results demonstrate that the per-directory setting of journaling modes in dajFS's design has sufficient validity.

5.2.2 MySQL

MySQL is also an open-source implementation of a relational database management system. It can create temporary files not only for maintaining data consistencies but also for gaining processing speed. For example, when processing an SQL statement that has "`ORDER BY`" or "`GROUP BY`", MySQL writes the result of a quick sort to a temporary file. Clearly, such a temporary file is less significant than database files.

We used dajFS and let MySQL create temporary files under a directory whose journaling mode was *writeback*. In addition, we set *journal* mode to the directory under which the database files were to be placed.

In these experiments, we measured the time it took to sort ten million randomly generated data by their dates in a table of integer, string, and date.

Figure 10 presents the results. For HDD, the elapsed time of dajFS was 12% shorter than that of ext3-d. Similarly, the improvement in elapsed time was about 30% for SSD.

These results demonstrate that the overhead of journaling temporary files in MySQL can be reduced by appropriately utilizing the per-directory journaling mode served by dajFS.

5.3 Reliability of File System

To determine if dajFS is really capable of maintaining the specified consistencies of the file system, we conducted experiments that intentionally make the operating system crash during file operations. To this end, we implemented a kernel thread that caused a kernel panic after a predetermined period of time when I/O operations for some target file were issued. As the target, we created a new 1 KB file under a directory whose journaling mode was *lightweight*, *writeback*, *ordered*, or *journal*. Then we performed 2 KB of write operations to the target file, and let the kernel thread cause kernel panic.

We classified the state of the file system into the following four categories.

State 1 The entire file system was not destroyed, i.e., both i-node bitmaps and data bitmaps are in normal state.

State 2 The target file was not destroyed, which means that its i-node is normal. This is the same desired consistency as that by ext3-w.

State 3 The content of the target file had no abnormality, which means that its i-node referred to the correct actual data. This is the same desired consistency as that by ext3-o.

State 4 The content of the target file was either the one before the write operations or the one after the write operations. This is the same desired consistency as that by ext3-j. In the latter case, the data block was written into the permanent storage safely.

Relationships between the journaling modes of dajFS and these states are as follows.

- The *lightweight* mode guarantees State 1.
- The *writeback* mode guarantees until State 2.
- The *ordered* mode guarantees until State 3.
- The *journal* mode guarantees until State 4.

After kernel panic caused by the kernel thread, we investigated the state of the file system and the target file by using a file system analysis tool called the Sleuth Kit [*7]. Results showed that dajFS

[*6] The default is "delete" mode.

[*7] https://www.sleuthkit.org/

succeeded in keeping the desired state, i.e., consistency, for every journaling mode of the parent directory of the target file.

6. Related Work

Okeanos [13] is a journaling file system that has two journaling modes: *wasteless* and *selective*. From the observation that write operations with small sizes of modifications can be a main cause of performance degradation of the system [14], the *wastelesss* mode commits multiple data caches simultaneously by combining their modified parts to form a single data, thus reducing the cost of I/O operations. To avoid meaningless page duplication, Okeanos also provides *selective* mode. In the *selective* mode, a write operation uses the *wasteless* mode when the size of the write operation is less than a certain threshold, and uses the *ordered* mode otherwise. Okeanos is implemented as an extension of ext3.

Adaptive Journaling [15] is a mechanism that automatically chooses appropriate journaling modes from the viewpoint of reducing waiting time for I/O operations on the basis of I/O patterns in transactions. It sets a journaling mode *per transaction*. Specifically, it uses *ordered* mode for transactions that perform sequential writes and uses *journal* mode for the other transactions to reduce the seeking time of a disk.

Journaling mode selection in both Okeanos and Adaptive Journaling is completely automatic, which means the user cannot specify desired journaling modes. In contrast, dajFS lets the user select which journaling mode to use for each directory.

File-adaptive Journaling [8] is a file system in which the user can set a journaling mode *per file*. As a result, similar to dajFS, more than one journaling modes co-exist in a file system. Although the user can set journaling modes in a finer manner than ext3, specifying the mode for every file is very time-consuming and puts an enormous burden on the user. In contrast, by using dajFS, the user can eliminate this burden. For example, the source of Linux kernel 4.2 contains M directories and N non-directory files, where $M = 3{,}376$ and $N = 50{,}781$ after extracting files from the archive, and $M = 6{,}677$ and $N = 110{,}319$ after making the kernel. Thus, in the file-adaptive journaling, the user sets desired journaling modes for at most N files. On the other hand, by using dajFS, the user has only to set desired journaling modes for at most M directories, provided that there are no problems in applying the same journaling mode to all files directly under each directory.

7. Conclusion

In this paper, we proposed dajFS, a journaling file system that offers per-directory journaling modes that can be changed without unmounting the file system. By using dajFS, the user can choose the appropriate journaling mode for a directory according to the importance of the files placed directly under that directory.

The experimental results demonstrate the effectiveness of dajFS. Although dajFS was implemented as an extension of ext3, it has almost no extra overhead compared to ext3. For SQLite, which guarantees the consistency of database files by its own mechanism, using dajFS reduced the execution times by up to 36%. For MySQL, which makes many temporary files, the execution times were reduced by up to 30% by using dajFS.

In future work, we will examine the correctness of dajFS by using model checking. We expect that a method proposed by Yang et al. [16] might be applicable to dajFS. We also want to adapt dajFS to a clustered file system. As suggested in the research by Hatzieleftheriou and Anastasiadis [17], extending the journaling mechanism of dajFS to a clustered file system could broaden the applicability of dajFS.

References

[1] Kowalski, T.J.: Fsck – The UNIX File System Check Program, *UNIX Vol. II*, Hume, A.G. and McIlroy, M.D. (Eds.), W.B. Saunders Company, pp.581–592 (1990).

[2] Hagmann, R.: Reimplementing the Cedar File System Using Logging and Group Commit, *Proc. 11th ACM Symposium on Operating Systems Principles*, *SOSP 1987*, pp.155–162 (1987).

[3] McKusick, M.K. and Ganger, G.R.: Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem, *Proc. FREENIX Track: 1999 USENIX Annual Technical Conference*, pp.1–17 (1999).

[4] Ganger, G.R., McKusick, M.K., Soules, C.A.N. and Patt, Y.N.: Soft Updates: A Solution to the Metadata Update Problem in File Systems, *ACM Trans. Comput. Syst.*, Vol.18, No.2, pp.127–153 (2000).

[5] Rosenblum, M. and Ousterhout, J.K.: The Design and Implementation of a Log-structured File System, *ACM Trans. Comput. Syst.*, Vol.10, No.1, pp.26–52 (1992).

[6] Sweeney, A., Doucette, D., Hu, W., Anderson, C., Nishimoto, M. and Peck, G.: Scalability in the XFS File System, *Proc. 1996 USENIX Annual Technical Conference*, *ATC 1996*, pp.1–14 (1996).

[7] Solomon, D.A. and Custer, H.: *Inside Windows NT*, Microsoft Press, 2nd edition (1998).

[8] Shen, K., Park, S. and Zhu, M.: Journaling of Journal is (Almost) Free, *Proc. 12th USENIX Conference on File and Storage Technologies*, *FAST 2014*, pp.287–293 (2014).

[9] McKusick, M.K., Joy, W.N., Leffler, S.J. and Fabry, R.S.: A Fast File System for UNIX, *ACM Trans. Comput. Syst.*, Vol.2, No.3, pp.181–197 (1984).

[10] Chidambaram, V., Pillai, T.S., Arpaci-Dusseau, A.C. and Arpaci-Dusseau, R.H.: Optimistic Crash Consistency, *Proc. 24th ACM Symposium on Operating Systems Principles*, *SOSP 2013*, pp.228–243 (2013).

[11] Lim, S.-H., Choi, H.J. and Park, D.-S.: Efficient Journaling Write-back Schemes for Reliable and High-performance Storage Systems, *Personal Ubiquitous Comput.*, Vol.17, No.8, pp.1761–1774 (2013).

[12] LSB Workgroup, The Linux Foundation: Filesystem Hierarchy Standard (2015), available from ⟨https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf⟩.

[13] Hatzieleftheriou, A. and Anastasiadis, S.V.: Okeanos: Wasteless Journaling for Fast and Reliable Multistream Storage, *Proc. 2011 USENIX Annual Technical Conference*, *ATC 2011*, pp.235–240 (2011).

[14] Nightingale, E.B., Veeraraghavan, K., Chen, P.M. and Flinn, J.: Rethink the Sync, *ACM Trans. Comput. Syst.*, Vol.26, No.3, pp.6:1–6:26 (2008).

[15] Prabhakaran, V., Arpaci-Dusseau, A.C. and Arpaci-Dusseau, R.H.: Analysis and Evolution of Journaling File Systems, *Proc. 2005 USENIX Annual Technical Conference*, *ATC 2005*, pp.105–120 (2005).

[16] Yang, J., Twohey, P., Engler, D. and Musuvathi, M.: Using Model Checking to Find Serious File System Errors, *ACM Trans. Comput. Syst.*, Vol.24, No.4, pp.393–423 (2006).

[17] Hatzieleftheriou, A. and Anastasiadis, S.V.: Host-side Filesystem Journaling for Durable Shared Storage, *Proc. 13th USENIX Conference on File and Storage Technologies*, *FAST 2015*, pp.59–66 (2015).

Wataru Aoyama received his M.E. degree from the University of Electro-Communications in 2018 and has been engaged in Fujitsu Limited since 2018. His research interests are operating systems and systems software.

Hideya Iwasaki is a Professor in the Graduate School of Informatics and Engineering at the University of Electro-communications. From 2011, he is a member of Science Council of Japan. He received his M.E. degree in 1985, his Dr. Eng. degree in 1988 from the University of Tokyo. His research interests are programming languages and systems, parallel processing, systems software, and constructive algorithmics. He is a member of the IPSJ and ACM.