Cleaning Suggestions for dtfs

Christian Czezatke

Oct, 28th. 1998

Introductory Remark

This documents contain a few comments and suggestions on how to integrate a cleaner with the existing dtfs module. Just consider it as an informal paper describing my thoughts on cleaning for dtfs.

Just a few basic considerations about cleaning in dtfs:

When implementing a cleaner for dtfs, care must be taken not to compromise the transaction-like semantic of disk accesses in a log-structured filesystem.

Cleaning a segment consists of the following steps:

1. Find a segment to be cleaned
2. Write all live blocks in this segment to the log
3. Wait until the log has commited the write triggered off in step 2
4. Mark the segment as free again.

Considering the transaction-like semantic of write accesses in a log-structured filesystem is is important to actually wait for the log write to finish before marking the segment as clean.

Finding a segment to be cleaned

Selecting an appropriate segment for cleaning is probably the most interesting task to be performed by the cleaner daemon. However, I'd start out with a simple algorithm (like trying to clean segments that are older than n logical time stamps compared to the last major checkpoint consisting of at least k percent of unreferenced blocks).

Write all live blocks to the log

For writing live blocks to the log by the cleaner, a new system call must probably be added. The problem with using the normal "write" function is that this updates the file's creation and modification time to the current time.

However, a straightforward approach should be to to use the implementation of "dext2_file_write" as it can be found in "dtfs/dext2/dext2_file.c" and just remove the line updating the ctime/atime field of the respective inode.

Wait until the log has commited the write

It should be possible to do this in the kernel module by simply issuing something like the following command sequence:

```
db_check_free_space(DEXT2_SB(super), 0, 0, 0);
de_finished(DEXT2_SB(super), CTYPE_MINOR);
sync_buffers(DEXT2_SB(super)->device, 1);
```

Furthermore, by using this sequence, you don't have to worry about possible race conditions. They are handled just ok by this calling sequence.

Mark the segment as free again

Segments in the log form a doubly linked list in dtfs (as in Sprite and 4.4BSD LFS). So if a segment gets cleaned, it must be removed from the list. This requires the segment header of the previous and the next segment to be updated, too. Again, care must be taken not to compromise the transaction-like nature of a log-structured filesystem.

This could be done using the following steps (The segment to be marked as free again is s_n, whereas s_{n-1} stands for the previous segment in the log and s_{n+1} indicates the next segment in the log:

1. Mark s_n as segment to be cleaned (by setting a SEG_IN_CLEANING flag in the segment header, for example)
2. Make the "next-pointer" in s_{n-1} point to s_{n+1} instead of s_n.
3. Make the "prev-pointer" in s_{n+1} point to s_{n-1} instead of s_n.
4. Update the segment usage bitmap to list s_n as free again.
5. Remove the SEG_IN_CLEANING flag in the header of s_n and mark the segment as clean again.

Performing freeing a segment this way should make it possible to recover the filesystem even if a crash occurs between any of the steps listed above.

Steps 1-4 can be considered as atomic since they only involve writing one block to the device.

Step 5 is made atomic by using the segment bitmap checksum. (Currently not implemented in the dtfs kernel module). If the checksum matches, all blocks in the segment usage bitmap are up to date.

Tips and Caveats

- You'll probably have to obtain a description for all the blocks in a segment. You might find having a look at "dtfslib" and the "inspect" utility as useful. dtfslib still has a lot of functionality to do things like locating the header of a segment quite easy. Just have a look at the sources of dtfslib and "inspect".
- Looking at dtfs filesystems with "inspect" might provide some help in understanding the dtfs data structures.
- Don't clean segments containing a "live" major checkpoint.

  With the current state of dtfs this translates into "don't clean segments that are pointed to in one of the two checkpoint areas as containing major checkpoints". If you don't adhere to this, a future fsck utility will have a hard time in reconstructing a filesystem that has not been unmounted cleanly (i.e: the log does not end with a major checkpoint).
- Don't clean segments that have the SEG_LOCKED or the SEG_HAS_BADBLOCK flag set in their header.

  Locking segments might be useful if someone wants to be able to boot from a dtfs filesystem some day on the intel platform. Furthermore there might be some software out there that uses licensinc/copy protection mechanisms based on the physical position of some file on the disk (you'll never know what people will come up with one day...)

  Currently there is no way to use the SEG_LOCKED flag for something useful, but it would be nice if the cleaner would respect the flag... ;-)