Journaling FFS with WAPBL

Jörg Sonnenberger
joerg@NetBSD.org

Overview

- A short introduction to FFS
- WAPBL: Overview
- WAPBL: In-depth
- Performance
- Open issues
- Questions

A short introduction to FFS

- Superblock
- Inodes
- Directories
- Cylinder groups
- Consistency requirements

The FFS superblock

- Description of the filesystem
- Block size, fragment size, number of blocks, etc
- Time of last mount and if unmounted cleanly
- Summary of filesystem content
- Stored redundantly to protect against bad blocks etc
- Different versions, some fields added, some killed
- dumpfs(8) tells the version (FFSv2 for WAPBL!)

Inodes

- The file content, not the file name
- 128 Bytes for FFSv1, 256 Bytes for FFSv2
- Link count, time stamps, size, flags, ownership, ...
- References to the first 12 blocks and indirect blocks for the rest
- Last block can be partially allocated: fragments

- Not all blocks have to be allocated: holes
- Inodes never end with holes
- Extended Attribute block for FFSv2

Directories

- Records of inode number, record len, file type, name
- Padded to block boundaries
- "." and ".." as special entries

Cylinder groups

- Distribute files over disk, reducing fragmentation
- Contain fixed size inode lists
- Contain free space bitmaps
- Contain superblock copy

Consistency requirements

- Superblocks have to stay in sync
- Cylinder groups need consistent summaries and bitmaps
- Inodes must be freed once link count reaches 0
- Inodes must have indirect blocks written before writting the pointer
- Inodes must be initialized before creating directory entries
- Inode reference count must be modified on link(2) and unlink(2)

Practical example: mkdir(2)

- Allocate free inode
- Allocate block by marking it as used in the bitmap
- Write directory template with "." and ".." entry
- Increment reference count of parent directory
- Write inode to disk with allocated block referenced and ref count 2
- Write directory entry to parent directory
- Update statistics

WAPBL: Goals

- Crash recovery without fsck
- Improve performance by reducing synchronisation
- Potentially reduce number of disk seeks by allowing aggregation

- Simpler and less error prone than Soft Updates
- Trivial to use: mount -o log ...

WAPBL: Components

- The generic WAPBL backend
- Integration into FFS

Overview: The WAPBL backend

- Journal writing and replaying
- Journal records:
  - Block entry
  - Revocation of earlier journaled blocks
  - List of unreferenced allocated inodes
- bwrite / bdwrite registers buffer and defer writing

In-depth: Journal layout

- Circular buffer of records
- Header block at the start and the end of the log area
- Headers are written alternatively with generation counter
- Newer header determines newest valid and oldest active record
- Explicit disk synchronistation after all writes

In-depth: Journal layout (II)

- Block entries: to be written to given location after crash
- Block revocation: when changing from meta data to data block
- Unreferenced allocated inode:
  - During initialisation: mode = 0
  - Unlinked, but still open: mode != 0

In-depth: Journal replay

- Process all journal entries in order:
  - Block entries: add to hash table
  - Revocation entries: remove entries from hash table again
  - Unreferenced inodes: keep last entry
- If not mounting read-only, write all blocks back to disk
- Call filesystem backend for unreferenced inodes

- Shared code between kernel and fsck

Overview: FFS integration

- Journal location in superblock
- Registration of inode allocation and freeing
- Registration after freeing meta data blocks
- Annotate transaction borders
- Allocation of journal
- Journal replay on mount

Journal location

- End of partition:
  - Size limited only by disk space
  - Disk address, size and block size stored in superblock
- In-filesystem:
  - Limited to size of cylinder group
  - Address, size, block size and inode number in superblock
- On mount, journal is created on-demand:
  - At the end, if enough free space (1MB journal per 1GB size)
  - Inside the filesystem (up to 64MB, at least 1MB)

In-depth: mkdir(2)

- -> sys_mkdir
- -> ufs_mkdir
- Allocate and register new inode:
  ffs_valloc: UFS_WAPBL_BEGIN + ffs_nodealloccg + UFS_WAPBL_END
- UFS_WAPBL_BEGIN
- UFS_UPDATE -> unregister inode again
- (write template)
- UFS_WAPBL_END

In-depth: mkdir(2) journal record

- First transaction:
  - Cylinder group updates (Block entry)
  - Inode update (Block entry)
  - Unreferenced inode list
- Second transaction:
  - Inode update (Block entry)

- Inode update for parent (Block entry)
- Directory content (Block entry)
- Unreferenced inode list

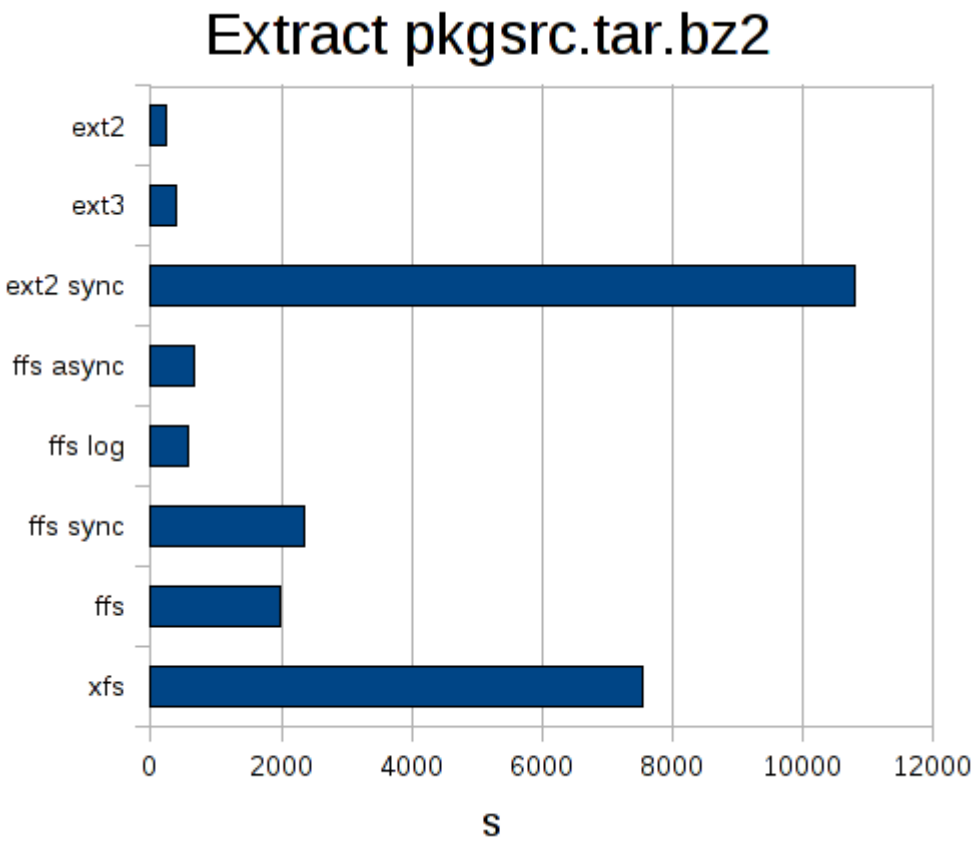
In-depth: ffs_write

- Can be called from inside the filesystem code or from sys_write/vn_write
- UFS_WAPBL_BEGIN if not already inside a transaction
- -> VOP_PUTPAGES
- UFS_WAPBL_END if started earlier

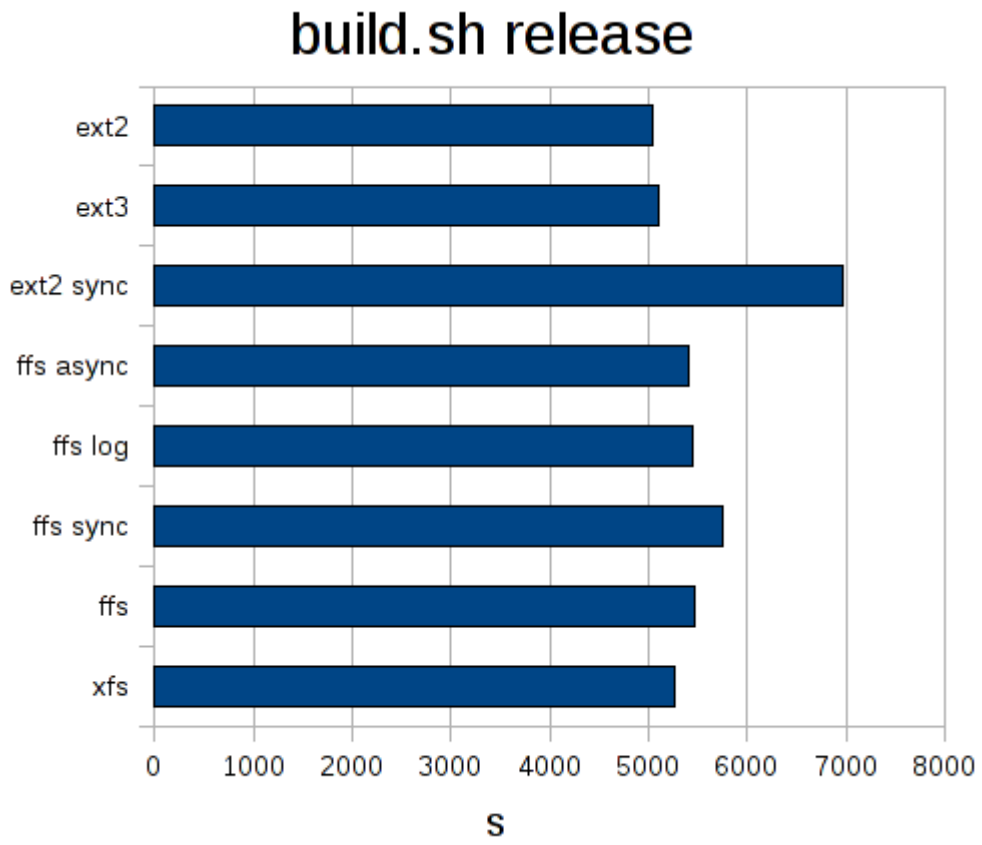
Performance: test system

- HP ProLiant ML110
- Xeon 3040 @1.86GHz
- 2GB memory
- Test on dedicated SATA disk, write caching enabled
- OpenSuSE 11.1 and NetBSD 5.0

Performance (I): 10x pkgsrc.tar.bz2



Performance (II): build.sh release



Open issues

- No checksum of journal entries
- Too much data flushing
- Too much serialisation of writes
- Holding the journal locked over UBC operations
- No data ordering
- Support for external journal

Q&A

Questions?