Storage Systems

NPTEL Course

Jan 2012

(Lecture 20)

K. Gopinath

Indian Institute of Science

Other Block Devices

- Network Block Devices
  - Instead of bus/SAN protocols, use IP networks
  - Higher rates of failure (non-availability or data loss)
    - Can use redundancy in software
  - Slower but more flexible (eg. use heterogenous devices)
  - Allows for much higher scaling
    - Bus/SAN networks cannot span continents
  - Parallelism an imp issue

Interaction of device and storage design

- Consider FAT filesystem
  - Linked list of clusters
- 2 file allocation tables (FATs)
  - Floppy devices very unreliable
- FAT systems on flash
  - Due to no update in place, every upd to a logical cluster in FAT will be written to a different physical cluster
  - May not need FAT2 at all!
  - Can we use an extra FAT to make it transactional?
- Transaction-Safe FAT File System (TFAT) (esp on flash)
  - driver layer modification to the original FAT file system
  - two file allocation tables (FAT0 and FAT1) but not identical
  - changes first made to FAT1
  - when transaction complete, FAT0 updated from FAT1, updating stable view of FS
  - However, not widely used! Only in mobile but not in desktop systems

F2FS: Flash friendly Filesystem

- Not for raw flash: assumes a FTL
  - jffs2, logfs for raw flash
- Based on a log structure file system
  - Segment (or region): 512 blocks of 4KB (2MB)
    - Many segments (2^k) make one section
  - copy-on-write: data always written to previously unused space
    - read from the most recently written region
    - mapping for reads changes for every update
  - free space managed in large regions that are written sequentially
  - cleaning: when number of free regions becomes low
    - live data coalesced from several regions into fewer regions and releasing rest
    - overhead is one of the significant costs of log structuring
- As NAND-based devices have different characteristics (due to internal geometry or FTL)
  - many parameters for configuring on-disk layout
  - for selecting allocation and cleaning algorithms

FTL and LFS

- FTL typically uses a log-structured design to provide wear-leveling and write-gathering
  - two log structures active on the device
  - f2fs uses FTL: f2fs makes no effort to distribute writes evenly to provide wear-leveling, as provided by FTL
  - f2fs provides large-scale write gathering: when many blocks need to be written at the same time, they are collected into large sequential writes that FTL can handle easily
  - But instead of a single large write, f2fs actually creates up to six in parallel.
    - Each set of blocks grouped with similar life expectancies
    - Makes garbage collection process required by the LFS less expensive
- However, f2fs doesn't always gather writes into large regions
  - Some metadata, and occasionally even some regular data, is written via random single-block writes: FTL takes over here
  - Simplifies design

Reducing Cleaning Overhead

- f2fs has six sections "open" for writing at any time
  - different types of data written to each section
  - different sections allows for file content (data) to be kept separate from indexing information (nodes),
  - Also to divide data into "hot", "warm", and "cold" sections (thru heuristics)
- Directory data treated as hot and kept separate from file data
  - have different life expectancies
- Section full of Cold Data likely to not require any cleaning
- Hot Nodes expected to be updated soon
  - if we wait a small amount of time, a section full of hot nodes will have very few live blocks: cheap to clean
- Problem: whenever a block written, its phys address changed, so its parent in the indexing tree must change and be relocated, and so on up to root of tree
  - Uses a special table for indirecting to actual blocks
  - Tree stores offset into table only; metadata changes do not need mod of tree
    - They indirect at the same offset in the table
  - Table needs updating
    - Table uses a special 2-location journaling to reduce overhead

Increasing Parallelism

- Many sections (1+) make a zone
  - Zones to try to keep the six open sections in different parts of the device
  - Assumption: flash devices often made from a number of fairly separate sub-devices each of which can process IO requests in parallel
  - If zones mapped to sub-devices, then the six open sections can all handle writes in parallel and make best use of device BW/minimize latency
- Zones the "main" area of the filesystem
- "meta" area contains a variety of different metadata
  - eg. segment summary blocks upd in place
  - This (small) area not managed by f2fs's lfs and left to FTL

Inode structure

- Uses standard Unix-like Inode
  - Indirects, also Double and Triple
  - Does not use B-Trees or extents
  - Inode size 4KB (larger than ext3) due to COW granularity
- Index tree for a given file has a fixed and known size
- when blocks relocated during cleaning, impossible for changes in available extents to cause indexing tree to get bigger
    - A problem as cleaning done to free space

Conclusions

- Nature of physical device has significant impact on design!