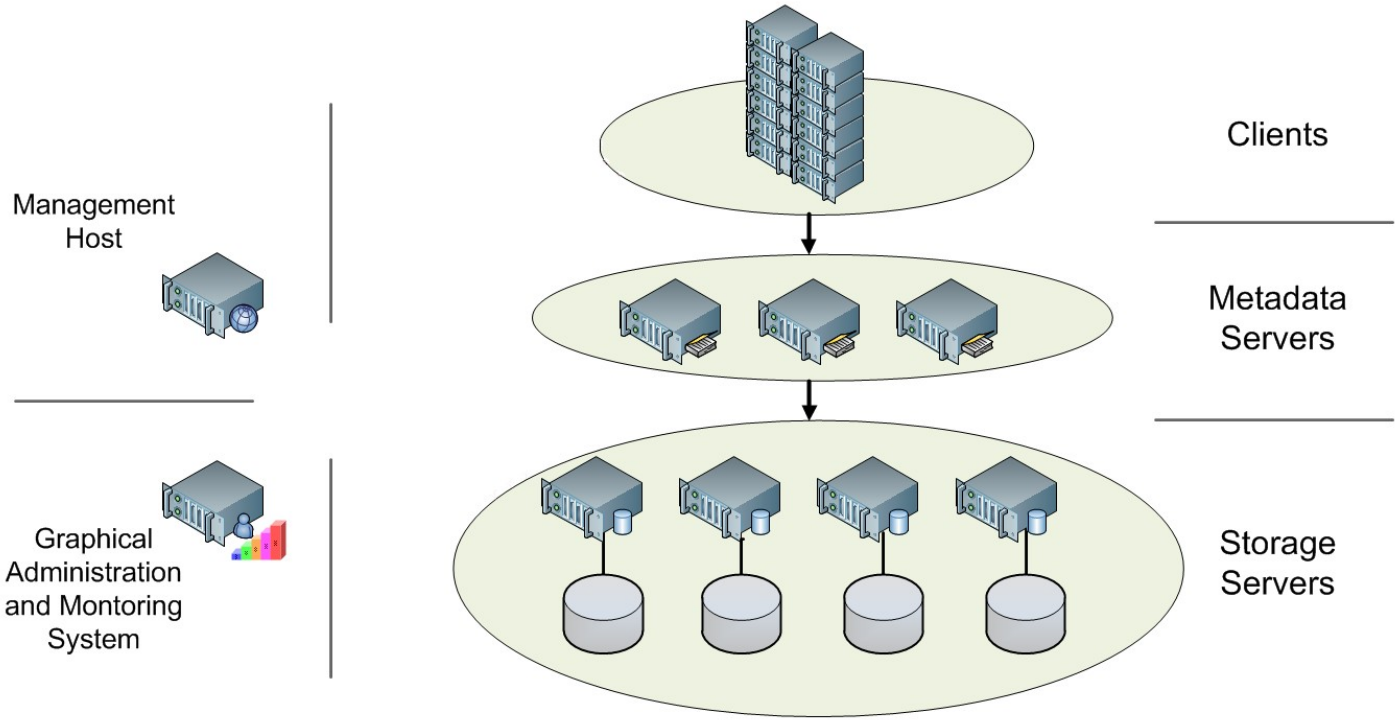
A dedicated cache for metadata?

Bernd Schubert
bernd.schubert@itwm.fraunhofer.de

Distributed file system



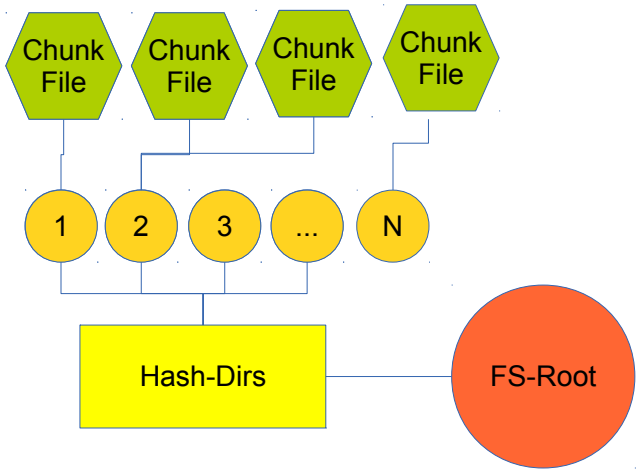
Distributed data and metadata

Hash directories (1)

- Meta and (object) storage servers of most distributed file systems use an existing underlying file system
- Using paths as the user sees it not optimal for data chunks (storage objects)
  - Meta operations such as rename, link, etc would need to be applied on storage servers as well
  - Especially difficult if storage server cannot be reached.
  - → Solution: path to chunk-file defined by an object ID
    - hash_dir = object-id % num_hash_dirs
    - Good (meta) performance with low number of files
- Hash directories also used for meta data by at least one distributed file system

Cache evictions

- Hashes cause random distribution of files between these directories
  - Random access
  - Directories (dir blocks) frequently evicted from cache
    - Data streaming
    - New files (inodes, dentries)



Performance

- Low meta performance (i.e. file creates, unlinks)
  - Single process creates/s down to 25/s (from above 2000/s) (8+2 raid6 with rotating disks)
  - Affects streaming as well due to meta disk seeks
- Affects streaming as well due to meta disk-seeks
  - Improved dir block caching by improving the (hash) path
    - i.e. parent-dir part of the path
    - Still not ideal if users choose to have many directories with few files only and random access between these directories.
  - Can be generalized to users accessing many directories
    - Not uncommon on file servers

Solution I - linux-3.11

- Situation in general got much better with 3.11, patch series from Mel Gorman included
  - "Obey mark_page_accessed hint given by file systems”
  - Low vfs_cache_pressure value now avoids directory block page evictions after io streaming
  - Rather magical parameter, difficult to judge which is the right value
    - Used in shrinker functions shrink_slab -> ... -> prune_super
    - Frees unused inodes and dentries
    - Per node patch in 3.12, but logic seems to be the same
    - Admin might want to control the number of cached inodes, dentries , directory blocks, bitmaps, extents, etc., separately

Solution II - dm-cache + ramdisk

- dm-cache
  - Use ram-disk as cache device
  - File systems now set REQ_MET as IO scheduler hint
  - Only use it to cache bio->bi_rw & REQ_MET
  - Patch dm-cache to cache only these bio's
  - Block based caching only, does not distinguish between different block types
    - Possible to introduce further REQ_MET_X flags
    - More flags would allow better analysis with blktrace
  - Somewhat hard to populate the cache
  - Designed for SSD devices
    - Btree layout, Journal device, etc
    - → More complex and probably slower than required

Solution II - dm-cache + ramdisk

- modprobe brd rd_nr=2 rd_size=4194304
- blksize=`blockdev --getsz /dev/md/storage`
- dmsetup create storage-cached --table "0 $blksize cache \
  /dev/ram0 /dev/ram1 /dev/md/storage 512 1 writethrough \
  default 0"

Solution III - Separately handled meta-cache?

- Cache meta disk blocks
  - Allows more fine grained meta cache tunings
    - File systems would need to use more REQ_META_X flags
  - On disk inodes much smaller than in-memory inodes
    - 256B to 512B vs. 1kiB to 2kiB
    - For most file systems and default inode size
  - Implementation: Add buffer-head rbtree to super-block
- Pre-reserve pages for metadata
  - Would allow to avoid the 'fight' between streaming io and meta-data
  - Implementation: Add list of free meta-pages to super-block

Solution III - Separately handled meta-cache?

- Cons
  - Only useful on file servers?
  - Takes additional memory
  - Introduces additional complexity
- Would such patches ever get accepted?
  - Probably at least needs to be optional

Solution IV - improve the shrinker

- Different inodes could be handled differently by the shrinker
  - Distinction between small and large files (in the sense of cached pages) and dir inodes
    - sb->s_{file,dir}_inode_lru, nr_unused_{file,dir} inodes
    - Add a per sb tunable to keep a number of unused dirs and (small-) files
      - Defaults to 0 to keep the old behavior
    - Define default 'small' and also make it tunable
    - Where to add these options?
      - Mount option, vfs ioctl, sysfs?

Questions? Comments?