Optimal Use of Filesystems

How to get the most out of current storage systems

Stephan Wiesand, DESY - DV -
IceCube Monte Carlo Meeting
Zeuthen, 2009-02-17

HELMHOLTZ | ASSOCIATION

DESY

How Current Bulk Filesystems Work

- typical:

1 Metatdata Server

- name of file/directory
- path
- owner/group
- permissions
- location/distribution

n >1 Storage Servers

- (chunks of) files

- client looks up files on the MDS
- bulk of data (file contents) flows between client and storage server
- assumptions:
  - much more data than metadata
  - many more data ops than metadata ops (lookup, open, create, remove, rename, ...)

DESY

Example: Lustre

- clients see a POSIX like filesystem

MDS/MGS

OSS with OSTs

- MDS: MetaData Server
  - an actual filesystem with *everything* but the actual data
- MGS: Management Server
- OSS: Object Storage Server
- OST: Object Storage Target

DESY

Example: Panasas

- clients see a POSIX like filesystem

Director Blade

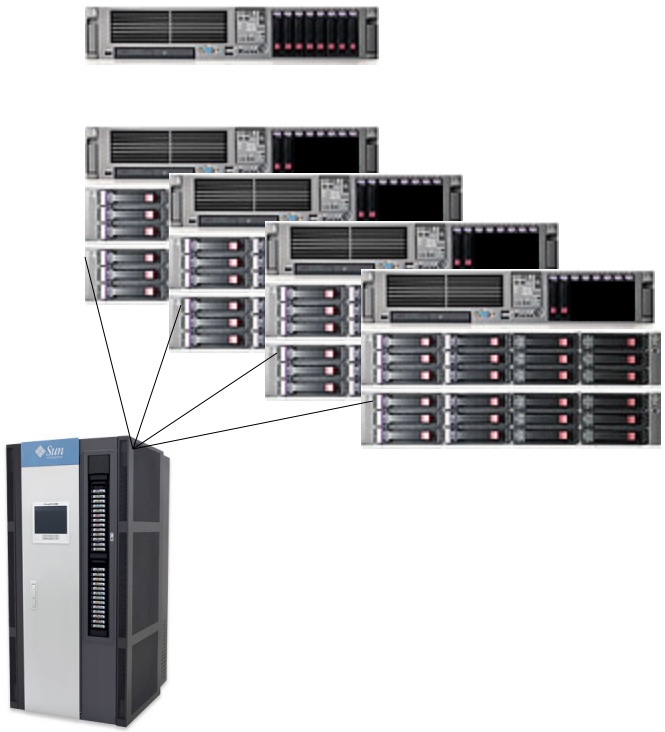
Storage Blades

- Director Blade: Volume and MetaData
  - service for a volume is located on exactly 1 director blade even if there are more
- Storage Blades: File Chunks

DESY

Example: dCache

- not POSIX like, started out as disk cache for files on tape



Head Node

Pool Nodes

- Head Node: database with metadata, location(s) on disk and on tape
- Pool Nodes: r/w, r/o, or w/o
- Optional tape backend

Common Characteristics

- Lustre, Panasas, dCache,... scale well for large files
  - scale with number of Storage Servers
  - capacity *and* throughput
- they don't scale so well for small files
  - metadata is a bottleneck, may limit rate of lookup/open/... operations
  - a 1kB file takes as much space on the MDS as on the Storage Server
  - => may even limit usable capacity
- => **keep files reasonably large**
  - at least on average
  - but preferably avoid small files at all
  - if you must keep several small auxiliary files per MC data file, at least tar them up

DESY

More Good Reasons for Large File Sizes

- tape backend:
  - *much* faster - acceptable speed only for files of at least O(100 MB)
  - also avoids “shoe-shining” (tape/drive wear -> data loss)
  - preferred size: as large as possible
  - but at DESY, still limited by 2GB size limit of tape backend software
- WAN transfers:
  - example: round trip time Zeuthen <-> Wisconsin: 136 ms
  - exchange of initial SYN/ACK packets takes at least this time
  - during this time, a 1 Gb link can transfer 16 MB of data
  - => no way to transfer files near wire speed unless they are >> 16 MB

Side Remark: AFS

- no scalability issues for small files

Volume Location Servers (clustered)

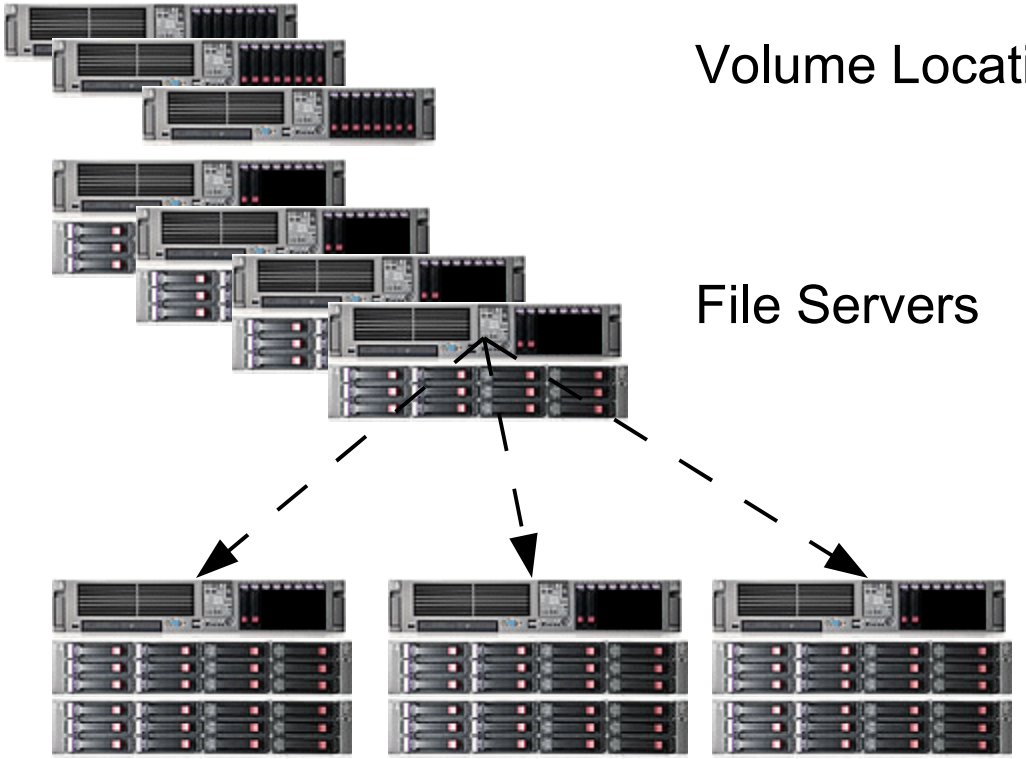
File Servers

- \+ File servers hold data *and* metadata for all files in a volume
- \- a volume is confined to a single partition on a single server
  - no automatic distribution of data from one volume

DESY

Side Remark: AFS+OSD (future)

- overcomes this AFS limitiation



Volume Location Servers (clustered)

File Servers

OSD Servers

- file server may store a file locally, as usual (small files)
- or delegate it to one ore more OSDs (large files)
  - clients talk directly to OSDs

Avoid Fragmentation

- affordable disk storage utilizes SATA drives
  - reasonable speed for streaming reads/writes
  - poor performance for anything else (seeks)
- => don't write too many files in parallel to the same filesystem
  - or data may not be contiguous on disk
  - Lustre with *n* OSTs: *n* writes are fairly safe (if OST capacity is balanced)
  - dCache with *n* pools: the same
  - if many more: possibly suboptimal read performance for these files *ever after*
- for the same reason: too many read streams hurt performance
- free space may also become fragmented
  - due to deletion of (small) files
- fragmentation becomes more problematic when filesystems fill up

Copying vs. Moving Files

- cp:
  - lookup file
  - read data from disk on source server, transfer to client
  - transfer data from client to destination server, write to disk
  - modify metadata to reflect new location
  - 3 x I/O as compared to writing to the right location in the first place
- mv:
  - lookup file
  - modify metadata to reflect new location
- => mv orders of magnitude cheaper, faster, and more scalable than cp
- alas, only works *within the same filesystem* or volume
- on the other hand, copying a file may defragment it...

Filename Length

- a directory is a list of files + corresponding metadata
- naive lookups are *O(n)* operations
  - can be really slow for long directories
- filesystems use hashing techniques to turn this into *O(log n)*
- alas, these hashes are built from the first *m* characters of the filename
  - `i3_mc_file_season_2008_par1_par2_par3_00000.data`
  - `i3_mc_file_season_2008_par1_par2_par3_00001.data`
  - `i3_mc_file_season_2008_par1_par2_par3_00002.data ...`
  - may all end up in the same bucket
  - and we're back to *O(n)*
- => keep filenames reasonably short

Locks

- don't use locks on shared filesystems

Summary: For Optimal Performance and Capacity...

- keep files large: O(100MB)+
- keep filenames short
- write data to the final destination in the first place
  - or at least move it around in the most efficient way
- try to avoid fragmentation
- don't overload the storage
- no locks
- use the right tool for the job
  - no filesystem is best for all purposes

DESY