Filesystem Storage Deduplication

IN4120 — Search Technology

Storage

- Modern capacities are still filled up
- Different applications, different needs
- Redundant backups
- Different users, identical data

Our goal

- **Find** duplicate files on live system
- Find duplicate **directories**
- **Avoid** insertion of already-existing files
- **Delete** unnecessary copies

Why bother?

Performance hits

- **Indexing** is not free
- Taking **backups** takes time
- **Virus** detection takes CPU time
- Overhead during **transactions**
- …

Economical costs

- Using **bandwidth**
- Upgrading storage
- …

More reasons: Data Management

Systems perspective

- Identifying original version
    - *“Fuck, I deleted the wrong version”*
    - Overhead during identification
        - Both for **computers** 🖥️
        - and our silly **brains** 🧠
- Validation

Data perspective

- Complexity during compression
- Entropy loss
- Two-phased (de)compression
- Copy-on-Write (CoW) latency

Benefits of free storage space

Operating system

- Swap space

File system

- Idle time tidying
  - Defragmentation
- Prefetching buffer

Identity comparison

- Early exit
- Hashing
  - Large files: preemptive comparison of fixed-size bytes
  - Small files: Bytewise?
- Metadata
  - Filesize
  - Filename?

- Reading from IO devices is SLOW
- Many-to-many comparisons

Comparing directories

Using hashes as content, recursively

- Destructive: loses meaningful information
    - Inability to do similarity comparisons

Metadata

- Number of files/directories
- Filetypes included

Top-bottom (breadth-first) vs Merkle tree

Dynamic systems

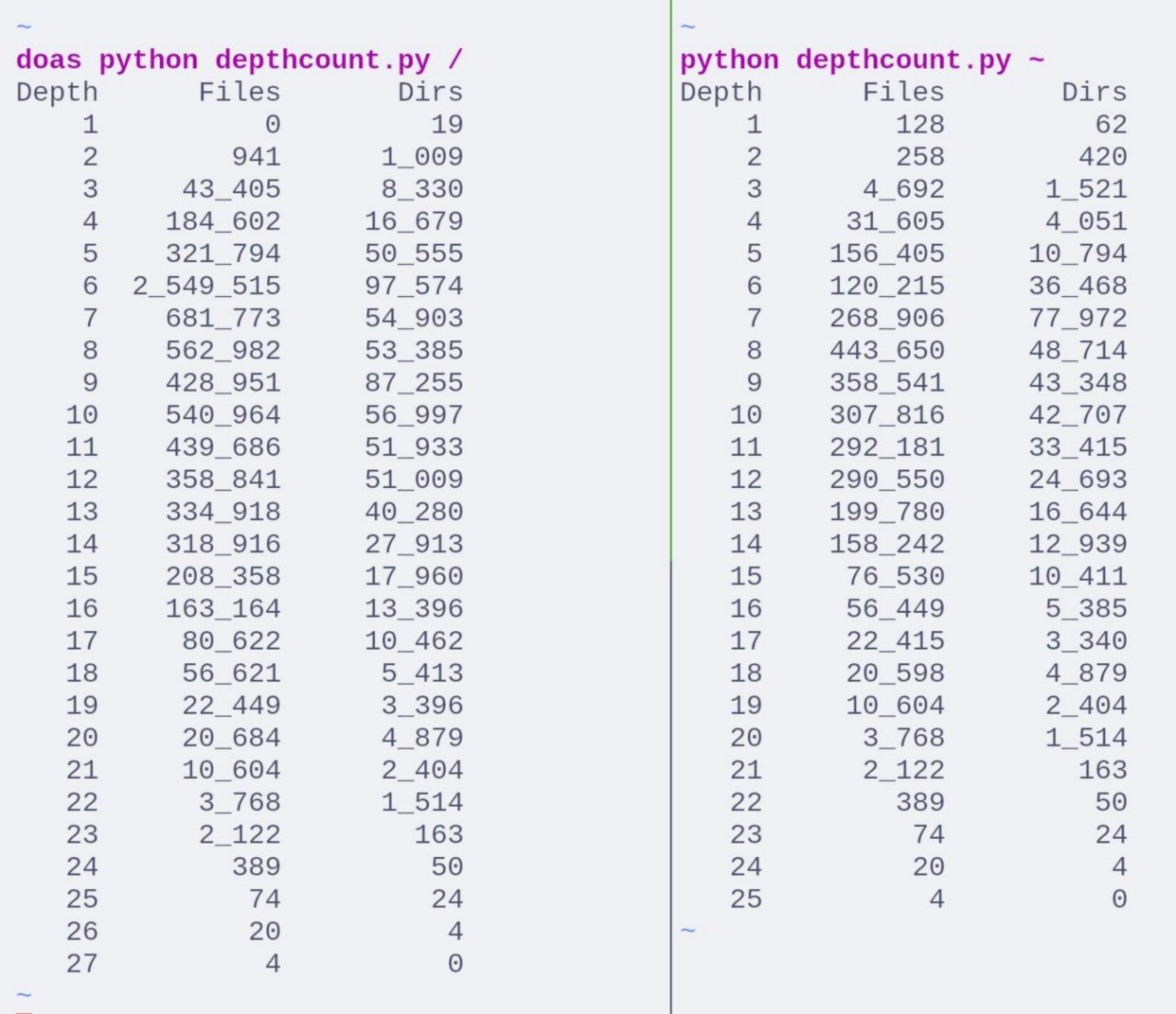
Bottom-up propagation approach

Merkle tree invalidation

- Invalidate parents on file change
- Avoid recomputing hashes
  - Blocking operation, adds write latency
  - May not be read before invalidated again

Filesystem support

- Hash information as xattr
- Computed during idle time
- Available from superblock metadata



```
~
doas python depthcount.py /
Depth        Files        Dirs
    1            0          19
    2          941       1_009
    3       43_405       8_330
    4      184_602      16_679
    5      321_794      50_555
    6    2_549_515      97_574
    7      681_773      54_903
    8      562_982      53_385
    9      428_951      87_255
   10      540_964      56_997
   11      439_686      51_933
   12      358_841      51_009
   13      334_918      40_280
   14      318_916      27_913
   15      208_358      17_960
   16      163_164      13_396
   17       80_622      10_462
   18       56_621       5_413
   19       22_449       3_396
   20       20_684       4_879
   21       10_604       2_404
   22        3_768       1_514
   23        2_122         163
   24          389          50
   25           74          24
   26           20           4
   27            4           0
~
```

```
~
python depthcount.py ~
Depth        Files        Dirs
    1          128          62
    2          258         420
    3        4_692       1_521
    4       31_605       4_051
    5      156_405      10_794
    6      120_215      36_468
    7      268_906      77_972
    8      443_650      48_714
    9      358_541      43_348
   10      307_816      42_707
   11      292_181      33_415
   12      290_550      24_693
   13      199_780      16_644
   14      158_242      12_939
   15       76_530      10_411
   16       56_449       5_385
   17       22_415       3_340
   18       20_598       4_879
   19       10_604       2_404
   20        3_768       1_514
   21        2_122         163
   22          389          50
   23           74          24
   24           20           4
   25            4           0
~
```

Probabilistic filters

- Inspired by Bloom filter

- Propagate up the tree structure
- Avoid recursion into deeper nesting levels
- Multiple hashing functions
    - Combine several metadata attributes
    - Different phases of filtering
        - Avoid work if not strictly necessary