

A presentation slide with a light blue header bar. The header contains the date 'March 19, 2012' on the left, the names '746 Spring 2012, Garth Gibson and Greg Ganger' in the center, and the number '1' on the right. The main content area is white. The title 'Project 2' is written in a large, bold, red font. Below it, the subtitle 'Hybrid SSD/HDD/Cloud Storage System' is written in a large, bold, black font.

March 19, 2012 748 Spring 2012: Garth Gibson and Greg Ganger 2

Project due on April 16th (11.59 EST)

- Start early ☺
 - Milestone 1: demo part 1 by March 28th
 - Milestone 2: demo part 2 by April 11th
 - Final Report Due: April 16th
- “Oh well, I will write the report on April 16th”
 - Bad idea ☹
 - 30% of the project grade is divided among project report and source code correctness/quality
 - Project report consists of design overview and performance analysis of your results

- March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 3
- Summary of the project
- Write a user-level file system called CloudFS that manages hybrid storage devices (SSD, HDD, and Cloud)
 - File system in user-space (FUSE) toolkit
 - Code to be written in C/C++
- Testing and evaluation setup
 - All development and testing done in Linux using the VirtualBox virtual machine setup
 - Your code must compile in the Linux images on the virtual machine

Part 0: Primer on the FUSE toolkit

- Interposition layer to redirect VFS calls to your user-level code
- FUSE client code is programmable
 - Talk to remote nodes
 - Interact with local FSs
- FUSE clients need to implement a minimal set of protocols

```
graph TD; Application[Application] --> VFS[VFS layer]; VFS --> Linux[Linux ext2]; VFS --> FUSE[FUSE module]; Linux --> Out[ ]; FUSE --> Client[user-level code (FUSE client)];
```

The diagram illustrates the FUSE architecture. It is divided into two sections by a horizontal dashed line. Above the line is the 'Application' box. Below the line is the 'VFS layer' box. An arrow points from 'Application' to 'VFS layer'. From 'VFS layer', two arrows branch out: one to a green 'Linux ext2' box and another to a 'FUSE module' box. An arrow points down from 'Linux ext2'. An arrow points from 'FUSE module' to a box labeled 'user-level code (FUSE client)'.

[illegible]

- March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 6
- Part 1: Hybrid Flash-Disk Systems
- Hybrid local storage systems: Best of both worlds
 - Capacity cost closer to magnetic disk
 - Random access speed closer to flash
- CloudFS is a layered file system
 - Higher-level file system (CloudFS) splits data between the two devices
 - Each device runs it's own local file system (Ext2)
 - you will (must) not modify Ext2
- Key idea
 - All small objects in flash, all large objects in magnetic disk
 - All small IO (metadata access) should go to the flash

March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 7

Size-based data-placement

- App does create(f1.txt)
- CloudFS creates "f1.txt" in SSD
- Ext2 on SSD returns a handle for "f1.txt" to FUSE
- FUSE "translates" that handle into another handle which is returned to the app
- App uses the returned handle to write to "f1.txt" on the SSD
- When "f1.txt" grows too big, CloudFS moves it to the HDD, and "f1.txt" on the SSD becomes a file that contains link to the file on HDD
- Because this migration has to be transparent, app continues to write as before (all writes go to the HDD).

Skeleton Code for CloudFS is provided to you in the distribution

March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 8

Attributes replication: avoid small IO on Disks

- After migration from SSD to HDD, replicate "real" attributes from HDD to SSD, to avoid doing small IOs on HDD
- Inode attributes are stored as (you choose):
 - (1) Extended attributes,
 - Or (2) customized list inside a file in SSD

March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 9

Logical and Physical View of CloudFS

March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 10

Testing and Evaluation

- Test scripts (test_part1.sh) that perform three steps
 - Extract an .tar.gz file in the CloudFS mount point
 - Compute checksum on all the files in the tar-ball
 - Perform a directory scan of all files in the tar-ball
- Script allows you to test with different dataset sizes
- To facilitate measurements, each test ...
 - ... will empty caches before runs by remounting CloudFS
 - ... will measure number of block IOs using "vmstat"
 - (virtualized setup makes time-based measurement hard)
- More details in the README file in the "src/scripts/"

March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 11

Expected output for correctness

- Test scripts returns the number of blocks read/written during each of the step (by parsing "vmstat -d")

Testing step	Expected Correct Output
Extracting a TAR file in CloudFS	For a small file, HDD should have zero blocks written
Performing a checksum on the all the files from the TAR file	For large files, HDDs should have a large number of blocks being read (compared to SSDs)
Scanning whole namespace of the TAR file using "ls -laR"	With attribute replication, only the SSD should have block reads (HDD should have zero reads)

- Other useful tool is btrace/blkttrace

March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 12

How to write the report?

- Key design ideas and data-structures
 - Pictures are useful; but good ones need thought and time (start early ☺)
- Reason about the performance
 - Don't just copy-paste the output into the report
 - Show us that you know why it is happening

Data set	# of blks read SSD	# of blks read HDD	# of blks written SSD	# of blks written HDD
d1	123	23	756	345
d2	144	0	101	1623

March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 13

Part 2: Cloud Storage System

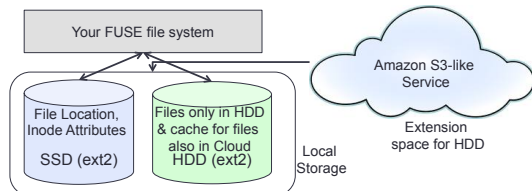
- Cloud Storage Services
 - Amazon S3, Dropbox, ...
- Advantages:
 - "Infinite" storage: supported by large data centers
 - Mobility: access anywhere you have Internet access
 - Sharing: easy sharing of some files with your family and friends
- Goal – Build your own cloud storage, using Amazon S3-like cloud storage service for capacity extension



March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 14

System Overview

- SSD manages metadata information on local machine:
 - File location: in SSD, HDD, or Cloud?
 - Namespace and inode attributes
- Cloud extends the capacity of local HDD
 - HDD is not a cache, because some files never go to cloud



March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 15

Amazon S3 storage model

- Object storage in flat namespace
 - Structure: S3://BucketName/ObjectName
 - (Bucket is a directory, object is a file)
 - List operations: look up the buckets
 - Put: write the data into S3
 - Get: read the data from S3
- Pricing (scaled down to meet our tests):
 - Capacity pricing: \$0.125 per MB (max size during one test)
 - Request pricing: \$0.01 per request
 - Data Transfer Pricing: \$0.120 per MB (Out from S3 only)

March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 16

Design decisions (and hints)

- Goal:
 - Expand the storage capacity by using cloud
 - Maintain good performance (low response time, HDD has what is needed most of the time)
 - Minimize cloud storage costs
- A simple example of caching:
 - LRU: Recently opened files should remain in local file system. When local file system is nearly full, least recently opened files are moved to cloud
 - Write-through: synchronize file to cloud when file is closed
 - Write-back: delay, possibly never, write to cloud

March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 17

Design decisions (and hints)

- Reduce capacity cost
 - Don't make copies of (all) files in cloud
 - Remove redundant data in the cloud
 - E.g. Windows desktop systems gain 75% storage savings by using block-level de-duplication
- Whole-file de-duplication:
 - If two files in cloud are the same, share only one copy (like hard links)
 - Not as effective as per-block, or the even better Rabin fingerprinting scheme [more in a later lecture]

March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 18

Whole-file De-duplication

- How to compare two files?
- Using compare-by-hash
 - If the hash of two files are the same, then are the two files are the same?
 - For n-bit hash function, it takes $\sqrt{2^{n+1} \ln(1/(1-p))}$ random files to find two files that have the same hash with probability p (known as the Birthday Paradox Problem)
 - SHA-1 has 160 bits, so $5.4 \cdot 10^{19}$ elements before collision (two different files with the same hash) with probability $< 1e-9$
 - SHA-256 has 256 bits, so $1.5 \cdot 10^{34}$ elements before collision with probability $< 1e-9$
 - It is **acceptable** to consider two files with same hash to be the same file (using SHA-n, $n \geq 1$)

Sharing files in Cloud

- Common characteristics of shared files:
 - E.g. music (mp3), photos (jpeg), and documents (pdf, pptx)
 - Embed key metadata inside the file
 - Other inode attributes (access mode) different for cloud sharing
- Sharing these files in well-known places
 - Four pre-defined types: music, movie, photo, and document
 - Translate pathname to the key name of objects in S3 ('/' → '+')
 - home/foo/my.mp3 → home+foo+my.mp3
 - No need for metadata (directory entries) from SSD
 - User sets extended attributes to instruct CloudFS to put a file in a specific place in the cloud:
 - `f = open("my.mp3"); setattr("my.mp3", "location", "cloud"); setattr("my.mp3", "type", "music"); ...`
 - Tells CloudFS to put "my.mp3" into your music folder in Amazon S3

Summary: data structures for CloudFS

- A list of data structures to maintain:
 - File Locations: URL identifies the location of a file in the cloud
 - Namespace (directory entries)
 - Inode attributes (timestamp, access mode ...)
 - LRU list: a list of objects in local storage sorted by last closed time
 - Hash value of file data for de-duplication
 - Reference count for each object
- DO's and DON'T's:
 - No in-memory solutions – information must survive crash/reboot
 - Out-of-core structures should use space on SSDs

Assumptions for simplification

- No user's file is larger than half of HDD capacity
- No need to store metadata in Cloud
- All metadata will fit into SSD (i.e. SSD is big enough)
- The capacity of HDD and SSD is passed as parameters
- Single threaded FUSE only is acceptable

Tools provided

- Amazon S3 client library:
 - Libs3: A C Library API for Amazon S3, a complete support of Amazon S3 based on HTTP protocols
 - Provide a wrapper of libs3 in CloudFS skeleton code, simplified synchronous call
- Amazon S3 server simulation:
 - A Python simulation run in VirtualBox
 - Implement simple APIs: list, put, get, delete
 - Store data in default local file system inside virtual box
 - Provide simple statistics about usage cost

Testing and Evaluation

- Correctness:
 - Basic functionality:
 - Read/Write files from cloud storage
 - Persistency: No data loss after normal umount/remount
 - Cache policy: LRU or any other advanced policy you invent
 - De-duplication: remove redundant contents
- Performance
 - Cloud storage usage costs
 - Local disk I/O traffics
 - CPU and memory usage

How to submit?

- Use Autolab for submission
 - Only test compilation for milestones
 - Real tests for grading are manually run with virtual box outside Autolab
- Deliverables:
 - Source code:
 - Good documentation in codes
 - Correct format for Makefile and input parameters
 - Follow instructions in handout to organize the code
 - Project reports
 - Key design ideas, data structures and reasons
 - Evaluation: local disk I/Os, cloud storage costs, etc.
 - No more than 4 pages, single column with 10 pts.

March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 25

Once again – start early ☺

- Project due on April 16th 2012
 - Milestone 1: finish part 1 by March 28th
 - Milestone 2: finish part 2 by April 11th
 - Leaves you a few days to work exclusively on writing final reports
- Demo
 - VirtualBox
 - FUSE
 - Running tests

March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 26

Back-up slides

March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 27

Analyzing statistics tools

- First, what is in the flash and disk ext2 file systems?
 - ls -allR /tmp/flash /tmp/disk
 - du -ak /tmp/flash /tmp/disk
 - Sorry for the lost+found side-effect of ext2 ☹
 - Look for the appropriate symbolic links and xattribute holding files
 - Look for the appropriate sizes of files (ext2 allocates 4KB blocks)
 - getfattr /tmp/flash/small/a/1 reports all attributes of file 1
- Next, what operations get done on the file systems?
 - In two separate terminal windows in Vbox, before your tests
 - [window1] btrace /dev/sdb1 | tee flashtrace
 - trace IOs to stdout (to watch) and file flashtrace (for later analysis)
 - [window2] btrace /dev/sdc1 | tee disktrace

March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 28

Btrace output

- dev, cpu, seq, time, pid, action, iotype, address+length, info

```

#16 0 1 0.000000000 23 A W #279 + 8 <- (8,17) #216
#17 0 2 0.000000844 23 Q W #279 + 8 [pdfiush]
#17 0 3 0.000012158 23 Q W #279 + 8 [pdfiush]
#17 0 4 0.000016528 23 P W [pdfiush]
#17 0 5 0.000018858 23 I W #279 + 8 [pdfiush]
#17 0 6 0.004114016 0 UPT W [swapper] 1
#17 0 7 0.006132063 10 D W [blklckd/0] 1
#17 0 8 0.004152279 10 D W #279 + 8 (kblockd/0)
#17 0 9 0.005084990 0 C W #279 + 8 [0]
#16 0 10 22.818093118 3234 A R 12375 + 8 <- (8,17) 12332
#17 0 11 22.818093658 3234 Q R 12375 + 8 [flashndisk]
#17 0 12 22.818105498 3234 Q R 12375 + 8 [flashndisk]
#17 0 13 22.818104768 3234 P W [flashndisk]
#17 0 14 22.818106018 3234 I R 12375 + 8 [flashndisk]
#17 0 15 22.818112428 3234 U W [flashndisk] 1
#17 0 16 22.818120222 3234 D R 12375 + 8 [flashndisk]
#17 0 17 22.818375663 3234 C R 12375 + 8 [0]
#16 0 18 22.818625434 3234 A R 20709447 + 8 <- (8,17) 20709384
#17 0 19 22.818625941 3234 Q R 20709447 + 8 [flashndisk]
#17 0 20 22.818627933 3234 Q R 20709447 + 8 [flashndisk]
#17 0 21 22.818627149 3234 P W [flashndisk]
#17 0 22 22.818629811 3234 I R 20709447 + 8 [flashndisk]
#17 0 23 22.818633398 3234 U W [flashndisk] 1
#17 0 24 22.818640422 3234 D R 20709447 + 8 [flashndisk]
#17 0 25 22.825372038 0 C R 20709447 + 8 [0]
    
```

March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 29

Understanding btrace output

- btrace is really blktrace | blkparse
- man blkparse tells you how to read the output
- As our devices are virtual, time is not very interesting
- We care about numbers of sectors read and written
- “Action C” is completion of an IO (address + length)
- Types are R read, W write, RA readahead
- Next you want to understand what is being read or written
 - need to tie “address” to disk structure
 - debugfs /dev/sdb1 # to debug ext2 file system on flash

March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 30

So the Analysis

- Can you attribute every sector read and written during your runs of md5 and ls on the flash, the disk?
 - Remember free list bitmaps for inodes and data blocks
 - Remember directory entries
 - Remember indirect blocks
 - Remember extended attributes (linked like indirect blocks)
 - Remember that inodes are smaller than blocks
 - Remember that “allocate, free, allocate” may be a new block
 - Accounting for everything may be hard, just try your best
- How well does this correspond to “small random on flash, large sequential on disk”?

March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 31

Test Case

- Functionality tests:
 - copy, untar, delete, calculate md5sum
 - Build simple projects
- Large file tests:
- Cache policy tests:
 - LRU and Write-Back Cache Policy
 - Generate LRU friendly access pattern
- De-duplication tests:
 - Generate several large files with the same contents
- Persistency tests:
 - umount / mount
 - Repeat the above tests to test performance difference

March 19, 2012 746 Spring 2012, Garth Gibson and Greg Ganger 32

Monitoring

- End-to-end running time
- SSD and HDD traffic:
 - Total number of read/write requests
 - Total number of read/write sectors
- Cloud storage:
 - Capacity usage
 - Total number of requests
 - Total bandwidth consumption
- CPU and memory usage