

ESE532: System-on-a-Chip Architecture

Day 16: October 27, 2021
Deduplication and Compression Project



Penn ESE532 Fall 2021 -- DeHon

Today

- Motivation (part 1)
- Project (part 2)
- Content-Defined Chunking (part 3)
- Hashing / Deduplication (part 4)
- LZW Compression Setup (part 5)

Penn ESE532 Fall 2021 -- DeHon

2

Message

- Can reduce data size by identifying and reducing redundancy
- Can
 - spend computation and data storage
 - to reduce communication traffic

Penn ESE532 Fall 2021 -- DeHon

3

Problem

- Always want more
 - Bandwidth
 - Storage space
- Carry data with me (phone, laptop)
- Backup laptop, phone data
 - Maybe over limited bandwidth links
- Never delete data
- Download movies, books, datasets
- Make most use of space, bw given

Penn ESE532 Fall 2021 -- DeHon

4

Opportunity

- Significant redundant content in our raw data streams (data storage)
- **More formally:**
 - Information content < raw data
- Reduce the data we need to send or store by identifying redundancies

Penn ESE532 Fall 2021 -- DeHon

5

Example

- Two identical files
 - Different parts of my file systems
- Don't store separate copies
 - Store one
 - And the other says "same as the first file"
 - e.g. keep a pointer

Penn ESE532 Fall 2021 -- DeHon

6

Why Identical?

- Eniac file system (common file server)
 - Multiple students have copies of assignment(s)
 - Snapshots (.snapshot)
 - Has copies of your directory an hour ago, days ago, weeks ago
 - ...but most of that data hasn't changed

Penn ESE532 Fall 2021 -- DeHon

7

Broadening

- History file systems
 - snapshot, Apple Time Machine
- Version Control (git, svn)
- Manually keep copies
- Download different software release versions
 - With many common files

Penn ESE532 Fall 2021 -- DeHon

8

Cloud Data Storage

- E.g. Drop Box, Google Drive, Apple Cloud
- Saves data for large class of people
 - Want to only store one copy of each
- Synchronize with local copy on phone/laptop
 - Only want to send one copy on update
 - Only want to send changes
 - Data not already known on other side
 - (or, send that data compactly by just naming it)

Penn ESE532 Fall 2021 -- DeHon

9

Functional Placement

- At file server or USB drive
 - Deduplicate/compress data as stored
- In client (laptop, phone)
 - Dedup/compress to send to server
- In data center network
 - Dedup/compress data to send between server
- Network infrastructure
 - Dedup/compress from central to regional server

Penn ESE532 Fall 2021 -- DeHon

10

Optimizing the Bottleneck

- Saving data (transmitted, stored)
- By spending compute cycles
 - And storage database
- When communication (storage) is the bottleneck
 - We're willing to spend computation to better utilize the bottleneck resource

Penn ESE532 Fall 2021 -- DeHon

11

Project

Part 1

Penn ESE532 Fall 2021 -- DeHon

12

Project

- Perform deduplication/compression at network speeds (1Gb/s)
- Use “chunks” instead of files
- Turn a raw/uncompressed data stream into one that exploits
 - Duplicate chunks
 - Redundancies within chunks

Penn ESE532 Fall 2021 -- DeHon

13

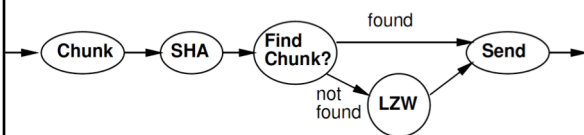
Project Context

- File server input link from network
 - Compress data before sending to disk
 - (or USB link from computer, compress before store to flash)
- Network link in data center or infrastructure
 - Compress data that goes over network

Penn ESE532 Fall 2021 -- DeHon

14

Project Task



Penn ESE532 Fall 2021 -- DeHon

15

Motivation

- Can we afford to simply compare every incoming file with all the files we've already sent?

Penn ESE532 Fall 2021 -- DeHon

16

Preclass 1

- How many comparisons per input byte in file?
 - Hint: how many total comparisons?

```

#define MAX_FILE_SIZE 4096
#define MAX_KNOWN_FILES (1024*1024)
#define -1
int find_file(char file[MAX_FILE_SIZE], int flen, char **known_files) {
    for(int i=0; i<MAX_KNOWN_FILES; i++) {
        bool match=true;
        for (int j=0; j<flen; j++) match=(match && (file[j]==known_files[i][j]));
        if (match) return(i);
    }
    return(NO_MATCH);
}
  
```

Penn ESE532 Fall 2021 -- DeHon

17

Requirements?

- Can we afford to simply compare every incoming file with all the files we've already sent?
- Data coming in at 1 GB/s
- Processor (or datapath) running at 1GHz
- How many comparisons needed per cycle with preclass 1 solution?
 - Hint: how many ns per input byte? Cycles?

Penn ESE532 Fall 2021 -- DeHon

18

Alternate Strategy

- Is there something we can compute on the input file that will let us
 - Know if a file is definitely not equivalent
 - So not worth checking every byte
 - Find the duplicate directly?

Penn ESE532 Fall 2021 -- DeHon

19

Alternatives

- How about
 - Look at size of file?
 - Look at 10 characters at fixed spots in the files?
 - E.g. bytes 11, 23, 113, 947, 1168,
- Could do better?
 - Could do something where changing any single character might be detected?

Penn ESE532 Fall 2021 -- DeHon

20

Exploring Alternatives

- What if we xor'ed together every byte in the file?
- What if we took sum of every word (group of 4 bytes) in the file?

Penn ESE532 Fall 2021 -- DeHon

21

Fingerprint, checksum, digest

- Compute a function on all the bytes in the file → **digest**
- Bins files into separate classes by the digest
 - Only need to check those
- As increase bits in digest
 - Make likelihood of two files having same digest smaller
- If can arrange for digests to essentially be unique – like a fingerprint

Penn ESE532 Fall 2021 -- DeHon

22

Hash

- A finite digest (fixed number of bits) computed on a potentially large collection of data (like a file)
- Ideally uniformly random digests
 - each hash value equally likely
- Use as building block for grouping and matching

Penn ESE532 Fall 2021 -- DeHon

23

Refined Strategy

- Keep a map of hash digests to files on the system
- On new file,
 - Compute hash digest on file
 - Only compare file contents against files with the same hash
- If hash is perfect with 20b, how does this reduce the number of files need to compare?

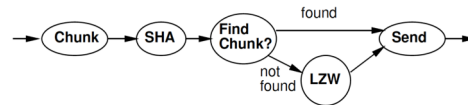
Penn ESE532 Fall 2021 -- DeHon

24

Hashing Impact

- With (perfectly distributed) k-bit hash
- $AvgSearch = \frac{TotalFiles}{2^k}$

Part 3: Content-Defined Chunking



Files or chunks?

- Why might files be the wrong granularity for identifying duplicates?

Blocks

- We regularly cut files into fixed-sized blocks
 - Disk sectors or blocks
 - inodes in File systems
- We could look for duplicates in blocks
- Why might fixed-sized blocks not be right division for deduplication?

Preclass 2 Unique Blocks?

Block 0	I am Sam. I am Sam. Sam-I-Am. That Sam-I-Am! That Sam-I-Am! I do not like that Sam-I-Am!	Block 4	I am Sam. I am Sam. Sam-I-Am. That Sam-I-Am! That Sam-I-Am! I do not like that Sam-I-Am! Do you like green eggs and ham?
Block 1	Do you like green eggs and ham? I do not like them, Sam-I-Am. I do not like green eggs and ham.	Block 5	I do not like them, Sam-I-Am. I do not like green eggs and ham. Would you like them here or there?
Block 2	Would you like them here or there? I would not like them here or there. I would not like them anywhere.	Block 6	I would not like them here or there. I would not like them anywhere. I do not like green eggs and ham.
Block 3	I do not like green eggs and ham. I do not like them, Sam-I-Am.	Block 7	I do not like them, Sam-I-Am.

Preclass 3 Unique Chunks?

Chunk 0	I am Sam. I am Sam. Sam-I-Am. That Sam-I-Am! That Sam-I-Am! I do not like that Sam-I-Am!	Chunk 4	I am Sam. I am Sam. Sam-I-Am. That Sam-I-Am! That Sam-I-Am! I do not like that Sam-I-Am!
Chunk 1	Do you like green eggs and ham? I do not like them, Sam-I-Am.	Chunk 5	Do you like green eggs and ham? I do not like them, Sam-I-Am.
Chunk 2	I do not like green eggs and ham. Would you like them here or there? I would not like them here or there. I would not like them anywhere.	Chunk 6	I do not like green eggs and ham. Would you like them here or there? I would not like them here or there. I would not like them anywhere.
Chunk 3	I do not like green eggs and ham. I do not like them, Sam-I-Am.	Chunk 7	I do not like green eggs and ham. I do not like them, Sam-I-Am.

Preclass 2 and 3

- Why are chunks able to do capture more duplicates?

Penn ESE532 Fall 2021 -- DeHon

31

Common File Modifications

- Add a line of text
- Remove a line of text
- Fix a typo
- Rewrite a paragraph
- Trim or compose a video sequence

Penn ESE532 Fall 2021 -- DeHon

32

Content-Define Chunking

- Would like to re-align pieces around unchanged/common sequences
 - Around the content
- Break up larger thing (file) into pieces based on features of content
 - Hence "content-defined"

Penn ESE532 Fall 2021 -- DeHon

33

Chunks

- Pieces of some larger file (data stream)
- Variable size
 - Over a limited range
- Discretion in how formed / divided

Penn ESE532 Fall 2021 -- DeHon

34

Chunk Creation

- How do we identify chunks?

Penn ESE532 Fall 2021 -- DeHon

35

Hashes and Chunk Creation

- Compute a hash on a window of values
 - Window: sequence of W-bytes
 - Like window filter

l i k e g r e e n e g g s a
0x20 0x67 0x72 0x65 0x65
0xC3

Penn ESE532 Fall 2021 -- DeHon

36

Hashes and Chunk Creation

- Compute a hash on a window of values
 - Window: sequence of W-bytes
 - Like window filter
- Scan window over the input

l i k e g r e e n e g g s a

0x20 0x67 0x72 0x65 0x65

0xC3

l i k e g r e e n e g g s a

0x20 0x67 0x72 0x65 0x65 0x63

Penn ESE532 Fall 2021 -- DeHon

0xC3 0x11

37

Hashes and Chunk Creation

- Compute a hash on a window of values
 - Window: sequence of W-bytes
 - Like window filter
- Scan window over the input
- When hash has some special value (like 0 or 0x11)

– Declare a chunk boundary

l i k e g r e e n e g g s a

0x20 0x67 0x72 0x65 0x65 0x63

Penn ESE532 Fall 2021 -- DeHon

0xC3 0x11

38

Hashes as Chunk Cut Points

- What does this do?
- Guarantees that each chunk begins (or ends) at some fixed hash
- For a particular substring that matches the target hash
 - Always occurs at beginning (or end) of chunk
- If have a large body of repeated text
 - Will synchronize cuts at the same points based on the content

Penn ESE532 Fall 2021 -- DeHon

39

Chunk Size

- Assume hash is uniformly random
- The likelihood of each window having a particular value is the same
- So, if hash has a range of N, the probability of a particular window having the magic “cut” value is 1/N
- ...making the average chunk size N
- So, we engineer chunk size by selecting the range of the hash we use
- E.g. 12b hash for $2^{12} = 4\text{KB}$ chunks

Penn ESE532 Fall 2021 -- DeHon

40

Chunking Design

- Raises questions
 - How big should chunks be?
 - Apply maximum and minimum size beyond content definition?
 - How big should hash window be?
- Discuss
 - What forces drive larger chunks, smaller?
 - How do large chunks help compression? Hurt?

Penn ESE532 Fall 2021 -- DeHon

41

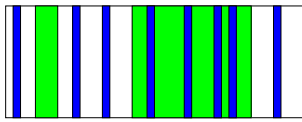
Example Text

- Consider beginning of repeated block of text.
- This stuff has already been seen.
- But, we are only matching on something that has a hash of zero.
- Maybe this line has a hash of zero.
- But, our repeated text is before and after the magic window with the matched hash value.

Penn ESE532 Fall 2021 -- DeHon

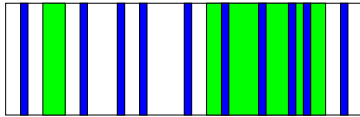
42

Example Data Stream



Blue: Hash=0.
Green: Identical

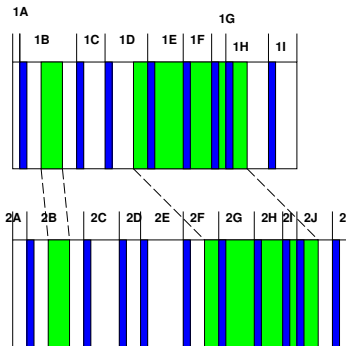
Maybe edited file,
added some content.



Penn ESE532 Fall 2021 -- DeHon

43

Example Data Stream



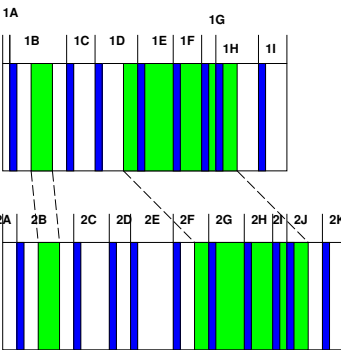
Blue: Hash=0.
Green: Identical

Maybe edited file,
added some content.

Penn ESE532 Fall 2021 -- DeHon

44

Example Data Stream



Blue: Hash=0.
Green: Identical

Maybe edited file,
added some content.

Which chunks are
exploitable duplicates?

1B - 2B ?
1D - 2F ?
1E - 2G ?
1F - 2H ?
1G - 2I ?
1H - 2J ?

Penn ESE532 Fall 2021 -- DeHon

45

Chunk Size

- Large chunks
 - Increase potential compression
 - ChunkSize/ChunkAddressBits
 - Decrease
 - Probability of finding whole chunk
 - Fraction of repeated content included completely inside chunks

Penn ESE532 Fall 2021 -- DeHon

46

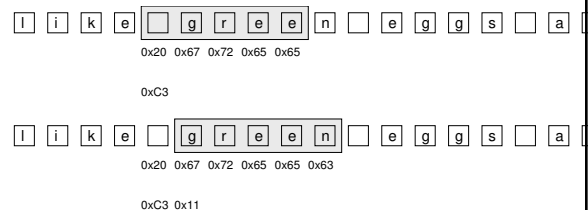
Rolling Hash

- A Windowed hash that can be computed incrementally
- $\text{Hash}(a[x+0], a[x+1], \dots, a[x+W-1]) = G(\text{Hash}(a[x-1], a[x+0], \dots, a[x+W-2])) - F(a[x-1]) + F(a[x+W-1])$
- i.e., hash computation is associative
- (+, - used abstractly here, could be in some other domain than modulo arithmetic)

Penn ESE532 Fall 2021 -- DeHon

47

Rolling Hash



- $\text{hash}(\text{gree}) = 0x20 + 0x67 + 0x72 + 0x65 + 0x65$
- $\text{hash}(\text{green}) = 0x67 + 0x72 + 0x65 + 0x65 + 0x6e$
- $\text{hash}(\text{green}) = \text{hash}(\text{gree}) - 0x20 + 0x6e$

Penn ESE532 Fall 2021 -- DeHon

48

Rabin Fingerprinting

- Particular scheme for *rolling hash* due to Michael Rabin based on polynomial over a finite field
- Commonly used for this chunking application

Penn ESE532 Fall 2021 -- DeHon

49

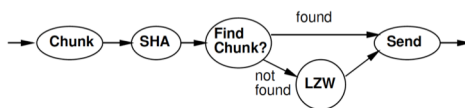
Content-Defined Chunking

- Compute rolling hash (Rabin Fingerprint) on input stream
- At points where hash value goes to 0, create a new chunk

Penn ESE532 Fall 2021 -- DeHon

50

Part 4: Hashing Deduplication



Penn ESE532 Fall 2021 -- DeHon

51

Hashes for Equality

- We can also (separately) take the hash signature of an entire chunk
- The longer we make the hash, the lower the likelihood two *different* chunks will have the same hash
- If hash is perfectly uniform,
 - N-bit hash, two chunks have a 2^{-N} chance of having the same hash.

Penn ESE532 Fall 2021 -- DeHon

52

Deduplicate

- Compute chunk hash
- Use chunk hash to lookup known chunks
 - Data already have on disk
 - Data already sent to destination, so destination will know
- If lookup yields a chunk with same hash
 - Check if actually equal (maybe)
- If chunks equal
 - Send (or save) pointer to existing chunk

Penn ESE532 Fall 2021 -- DeHon

53

Engineering Hash

- 2GB DRAM on Ultra96.
- How many 1KB chunks on a 1TB disk?
- Potential hash values for 256b hash?

Penn ESE532 Fall 2021 -- DeHon

54

Engineering Hash

- 2GB DRAM on Ultra96.
- $1\text{G} = 2^{30}$ 1KB chunks on a 1TB disk.
- 256b hash has 2^{256} potential hashes
 - Probably of same hash: 2^{-226}

Penn ESE532 Fall 2021 -- DeHon

55

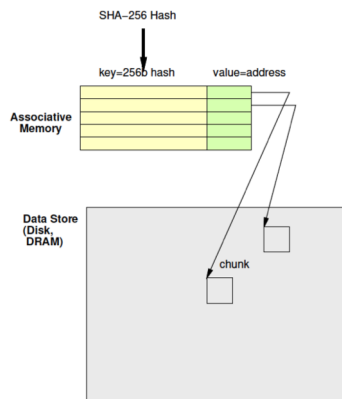
Deduplicate

- Compute chunk hash
- Use chunk hash to **lookup** known chunks
 - Data already have on disk
 - Data already sent to destination, so destination will know
- If lookup yields a chunk with same hash
 - Check if actually equal (maybe)
- How large of a memory do you need to hold the table of all 256b hash results?
- How relate to Ultra96 DRAM capacity?

Penn ESE532 Fall 2021 -- DeHon

56

Deduplication Architecture



Penn ESE532 Fall 2021 -- DeHon

57

Associative Memory

- Maps from a key to a value
- Key not necessarily dense
 - Contrast simple RAM
- Talk about options to implement next week

Penn ESE532 Fall 2021 -- DeHon

58

Secure Hash

- We regularly use digest signatures to identify if a file has been tampered with
- Again, hashes are same, mean data might be the same
- For security, we would like additional property
 - not easy to make the anti-tamper signature match

Penn ESE532 Fall 2021 -- DeHon

59

Cryptographic Hash

- One-way functions
- Easy to compute the hash
- Hard to invert
 - Ideally, only way to get back to input data is by brute force – try all possible inputs
- Key: someone cannot change the content (add a backdoor to code) and then change some further to get hash signature to match original

Penn ESE532 Fall 2021 -- DeHon

60

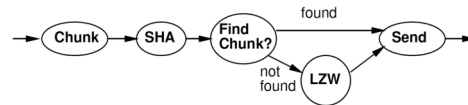
SHA-256

- Standard secure hash with a 256b hash digest signature
- Heavily analyzed
- Heavily used
 - TLS, SSL, PGP, Bitcoin, ...

Penn ESE532 Fall 2021 -- DeHon

61

Part 5: LZW Compression



Penn ESE532 Fall 2021 -- DeHon

62

Preclass 4

- I AM S<2,3><5,4><0,4>
- Message?

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
I		A	M		S											

Penn ESE532 Fall 2021 -- DeHon

63

Preclass 5, 6

- Bits in unencoded (decoded) message?
 - Assume 8b char
- Bits for encoded message?
 - Assume 9b for character
 - 1 bit to say is a character, then 8b char
 - And 9b for <x,y> pair
 - 1 bit char, 4b for each of x and y

Penn ESE532 Fall 2021 -- DeHon

64

Idea

- Use data already sent as the dictionary
 - Give short names to things in dictionary
 - Don't need to pre-arrange dictionary
 - Adapt to common phrases/idioms in a particular document

Penn ESE532 Fall 2021 -- DeHon

65

Encoding

- Greedy simplification
 - Encode by successively selecting the longest match between the head of the remaining string to send and the current window

Penn ESE532 Fall 2021 -- DeHon

66

Algorithm Concept

- While data to send
 - Find largest match in window of data sent
 - If length too small (length=1)
 - Send character
 - Else
 - Send $\langle x, y \rangle = \langle \text{match-pos}, \text{length} \rangle$
 - Add data encoded into sent window

Penn ESE532 Fall 2021 -- DeHon

67

Preclass 7

- How many comparisons per invocation?

```
#define DICT_SIZE 4096
#define LENGTH 256
// clen <= LENGTH
int longest_match(char dict[DICT_SIZE], char candidate[LENGTH], int clen) {
    int best_len=0; best_loc=-1;
    for (int i=0; i<DICT_SIZE-clen; i++) {
        j=0;
        while((candidate[j]==dict[i+j]) & (j<clen)) j++;
        if (j>best_len) {best_len=j; best_loc=i;}
    }
    return((best_loc<<8)|best_len);
}
```

Penn ESE532 Fall 2021 -- DeHon

68

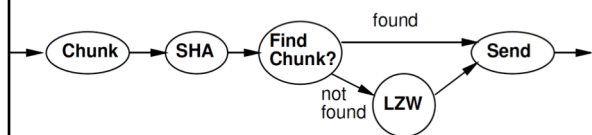
Next Time

- See a clever way to reduce comparisons to constant work per input character (linear in data being compressed)

Penn ESE532 Fall 2021 -- DeHon

69

Project Task



Penn ESE532 Fall 2021 -- DeHon

70

Big Ideas

- Can reduce data size by identifying and reducing redundancy
- Can spend computation and data storage to reduce communication traffic

Penn ESE532 Fall 2021 -- DeHon

71

Admin

- Feedback
- HW7 due Friday
- Project assignment out
- Reading for Monday online
- First project milestone due next Friday
 - Including teaming
 - Teams of 3

Penn ESE532 Fall 2021 -- DeHon

72