Final assignment of
"Management and Analysis of Physics Datasets" - Part 2

6th of July 2020

Introduction

This assignment covers three topics:

1. redundancy
2. cryptography
3. cloud storage technology

The first two assignments require some programming in a language of your choice. If you struggle to do this, please read the **hints** in both sections, which allow you to download the basic implementation in C/C++ and you can just add logic, which has to be added.

If you have any question to the exercise or find a mistake, send me an email to andreas.joachim.peters@cern.ch

1 Redundancy

We are programming a file based RAID-4 software algorithm. For this purpose we are converting a single input (**raid4.input**) file into four data files `raid4.0,raid4.1,raid4.2,raid4.3` and one parity file `raid4.4` – the four data and one parity file we call 'stripe files'.

The input file can be downloaded from:
http://apeters.web.cern.ch/apeters/pd2020/raid4.input

To do this we are reading in a loop sequentially blocks of four bytes from the input file until the whole file is read:

- in each loop we write one of the four read bytes round-robin to each data file, compute the parity of the four input bytes and write the result into the fifth parity file. (see the drawing for better understanding)

- we continue until all input data has been read. If the last bytes read from the input file are not filling four bytes, we consider the missing bytes as zero for the parity computation.

Input File (horizontal)

raid4.input - total size 170619 bytes
(number in cell = byte offset in file)

0	1	2	3	4	5	6	7	8	9	10	11	12	..	..		170618

Output Files (vertical)

(number in cell = byte offset in original file, p*0,1,2...* are the row-wise parities)

raid4.0	raid4.1	raid4.2	raid4.3	raid4.4
0	1	2	3	p0
4	5	6	7	p1
8	9	10	11	p2
12	13	14	15	p3
...	...	...	...	...

Stripe parity (column wise parity)

q0=0^4^8^12	q1	q2	q3	q4

Assignment 1

1.1 Write a program (C,C++ or Python), which produces **four striped data and one parity file** as described above using the given input file.
hint: *if you have a problem programming this yourself, you can download the core program in C++ from*
http://apeters.web.cern.ch/apeters/pd2020/raid4.c
See the explanations in the beginning how to compile and run it.
You have to add the parity computations at the IMPLEMENT THIS sections! If you can't compile or run it, you can still fill in the missing implementation!

1.2 Extend the program to compute additionally the parity of all bytes within one stripe file. You can say, **that the computed column-wise parity acts as a _____ for each stripe file**. Compute the size overhead by comparing the size of all 5 stripe files with the original file. **The size overhead is _____ %** !

1.3 What is the 5-byte parity value if you write it it in hexadecimal format like P^5 **`=0x[q0][q1][q2][q3][q4]`**, where the **[qx]** are the hexadecimal parity bytes computed by *xor-ing* all bytes in each stripe file. A byte in hexadecimal has *two digits* and you should add **leading 0** if necessary.
E*xamples*
- a byte with contents **1** in hexadecimal is **0x01.** A byte with contents **255** in hexadecimal is **0xff.**
- a possible 5-byte parity would be $\mathbf{P^5}$ = 0 x 01 0c 1a 2f 3e

1.4 If you create a sixth stripe file, which contains the row-wise parities of the five stripe files, **what would be the contents of this file?**

Write down the equation for R, which is the XOR between all data stripes D0,D1,D2,D3 and the parity P. Remember P was the parity of D0,D1,D2,D3! Reduce the equation removing P from it to get the answer about the contents!

1.5 After some time you recompute the 5-byte parity value as in 1.3. Now the result is $\mathbf{P^5}$ **= 0x ff 07 a0 9b 99**. Something has been corrupted. You want to reconstruct the original file raid4.input using the 5 stripe files. **Describe how** you can recreate the original data file. **Which stripe files do you use** and how do you recreate the original data file? **Why** could it be useful to **store** also the **file size** somewhere?

2 Cryptography

A friend has emailed you the following text: **`K]amua!kv$huvt`**

She told you that her encryption algorithm works like this:

- to each ASCI value of each letter I add a secret *key* value.
  (note that ASCII values range from 0 to 255)

- additionally to make it more secure I add a variable (so called) *nonce* value to each ASCII number.
  The *nonce* start value is 0 for the first character of the message. For each following character I increase the *nonce* by 1, e.g. for the second letter the *nonce* added is 1, for the third letter it is 2 aso.

```
encoded_character[i] = character[i] + key + nonce(i)
```

Assignment 2

2.1 Is this symmetric or asymmetric encryption and explain why?

2.2 Write a small *brute force* program which tests keys from 0..255 and use a dictionary approach to figure out the original message.

What is the decryption algorithm/formula to be used?

The **used key** is _____ , the **original message** text is _____ !

hint: *if you have a problem programming this yourself, you can download the core program in C++ from*
http://apeters.web.cern.ch/apeters/pd2020/decrypt.c

See the explanations in the beginning how to compile and run it.

You have to add the decryption formula at the IMPLEMENT THIS sections! If you can't compile or run it, you can still fill in the missing implementation!

3 Cloud Storage

In a cloud storage system we are mapping objects by name to locations using a hash table.

Imagine we have a system with ten hard disks (10 locations). We enumerate the location of a file using an index of the hard disk [0..9].

Example:
name=>location
---
file1=> 0
file2=> 2
file3=> 5

Our hash algorithm for placement produces hashes, which are distributed uniform over the value space for a flat input key distribution.

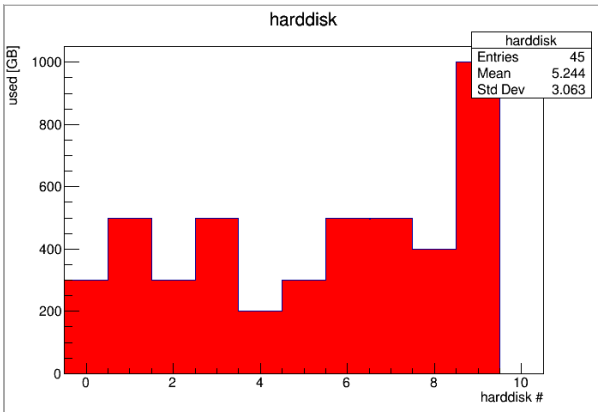
We want now to **simulate the behaviour of our hash algorithm without the need to actually compute any hash value.**

Instead of using real filenames, which we would hash and map using a hash table to a location (as we did in the exercise), we are ‘computing’ a **location** for ‘any’ file by **generating a random number for the location** in the range [0..9] to assign a file location. To place a file in the storage system we use this random location where the file will be stored and consumes space.

Assignment 3

Assume each disk has 1TB of space, we have 10TB in total.

Place as many files of 10GB size as possible to hard disks choosing random locations **until one hard disk is full.**
Hint: a hard disk is full once you have stored hundred 10GB files.



3.1 Write a program in C, C++, Python or using ROOT, which simulates the placement of 10GB files to random locations and account the used space on each hard disk. Once the first hard disk is full, you stop to place files.

Remark: the distribution changes every time if the random generator is not seeded always with the same start value. Nevertheless both ways are accepted!

Possibly **visualise the distribution** similar to the histogram above.

3.1a How many files did you manage to place?
3.1b What is the **percentage of total used space** on all hard disks in the moment the first disk is full?

3.2 Repeat the same task placing 1GB files until the first hard disk is full.
3.2a How many files did you manage to place?
3.2b What is the **percentage of total used space** on all hard disks in the moment the first disk is full?

3.3 Based on this observation: why do think cloud storage typically stores fixed size blocks of 4M and not files of GBs size as a whole?
(so called block storage approach)