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

1st of June 2021

Dr. Andreas-Joachim Peters - CERN

Introduction

This assignment covers the following topics:

1. **Redundancy**
2. **Cryptography**
3. **Object Storage Technology**
4. **REST APIs and Block Chain Technology**

The assigned topics require short programming in a language of your choice: Python (native or Notebook), R, GO, C, C++ aso….

Simple, clear and short programming solutions are always preferred and rewarded!

While in case of a more general question, don't write only one sentence with five words, elaborate a little bit the given subject!

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

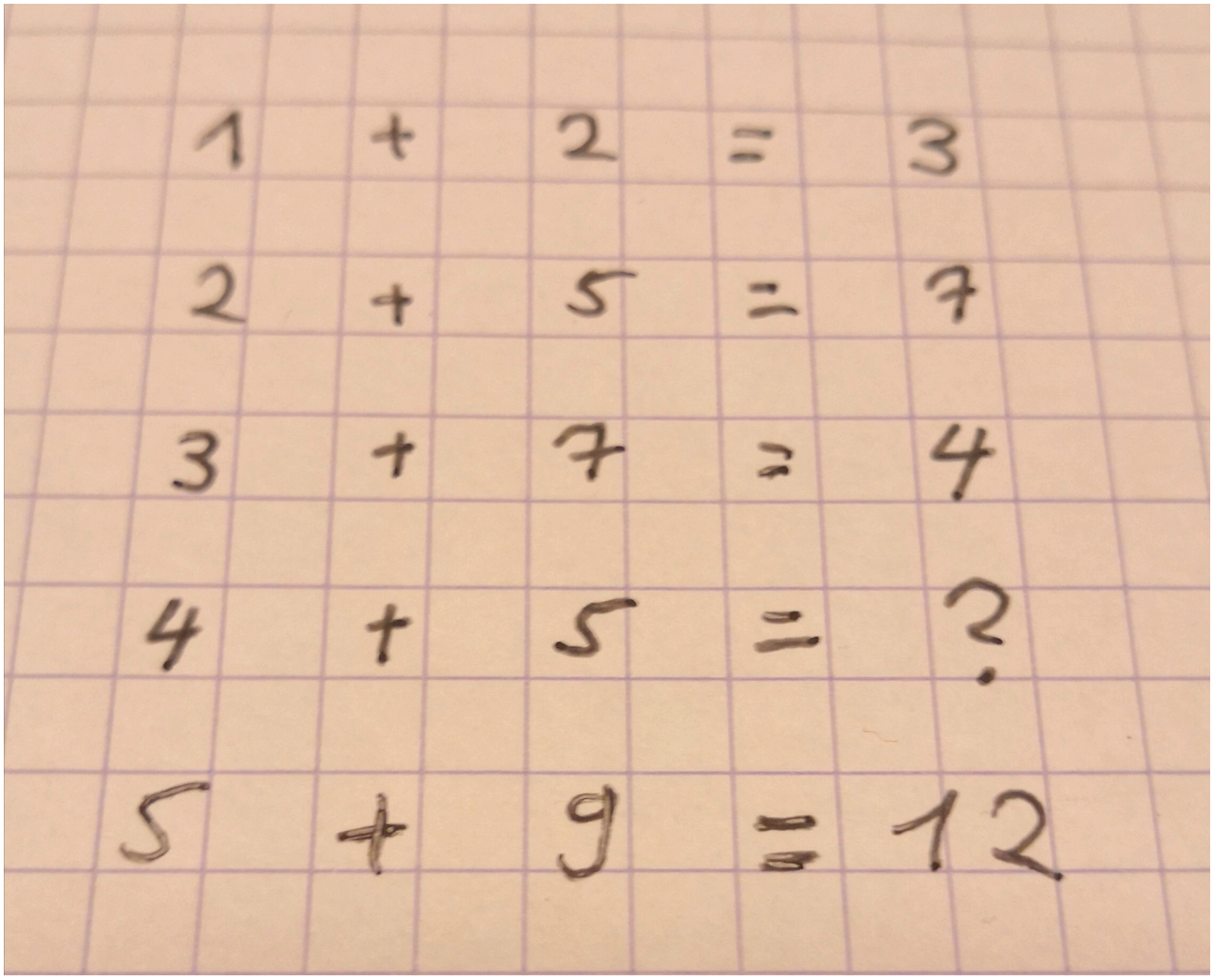
You hand in your assignments **via Email** before your chosen exam date in form of a **Notebook file + PDF printout** of the notebook - or as a **PDF** including **answers** and **source code.**

mail-to: andreas.joachim.peters@cern.ch

Links to GIT repositories or a notebook file without PDF printout are not accepted!

You are invited to return the assignment within a short time, even if you chose to take the exam at a later point.

The assignment might get modified directly after the first or second exam date if the authenticity and ownership of solutions seems not anymore given!



Fun Exercise

You might have seen this kind of puzzles on social media. It should be straight-forward for you to understand which operations connect the two numbers before the = sign.What is the result of the ? field ? Show proof for each line!

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/pd2021/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++, R 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/pd2021/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!

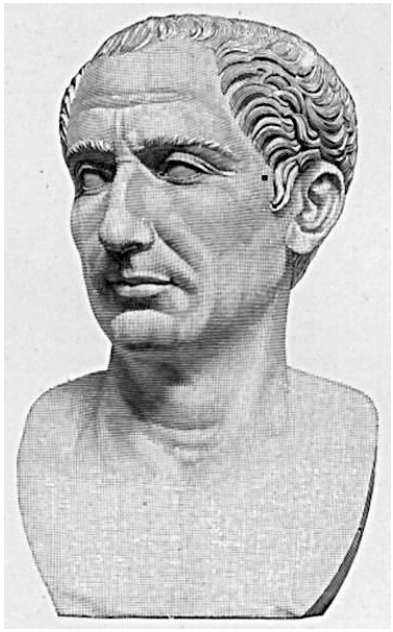
1.2 Extend the program to compute additionally the parity of all bytes within one stripe file. You can say, **that the computed column-wise parity acts as a _____ for each stripe file**. Compute the size overhead by comparing the size of all 5 stripe files with the original file. **The size overhead 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 a5 07 a0 01 9e**. 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 with the correct size?



2 Cryptography

The ***Caesar cipher*** is named for Julius Caesar, who used an alphabet where decrypting would shift three letters to the left.

A friend has emailed you the following text:

`K]amua!trgpy` Diverso da traccia vecchia

She told you that her encryption algorithm works similar to the Caesar cipher:

- 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.

  Vecchia traccia: 0

  The *nonce* start value is 5 for the first character of the message. For each following character add 1 to the *nonce* of the previous character, e.g. for the second letter the *nonce* added is 6, for the third letter it is 7 aso.

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. In Python you can use the *ord()* function to get an integer representation of a character and the *chr()* to retrieve a character string from an integer!

What is the decryption algorithm/formula to be used?

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

3 Object Storage

In an object 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.

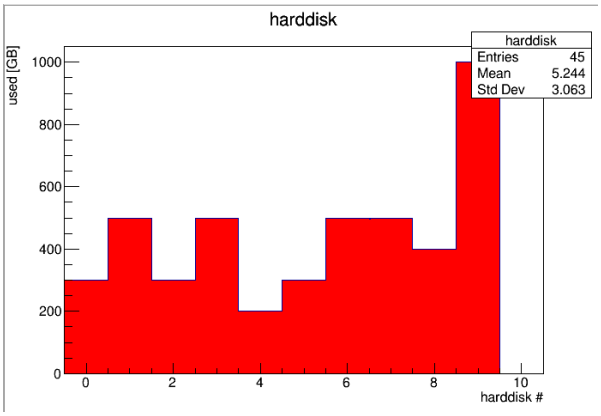
We want now to **simulate the behaviour of our hash algorithm without the need to actually compute any hash value.**

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 Python, R 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 you think object storage typically stores fixed size blocks of 4M and not files of GBs size as a whole?
(so called block storage approach)
Run the same program for 4M block sizes and demonstrate the benefits

3.4. Compute the **average** used space on all hard disks and the **standard deviation** for the average used space for 10 GB and 1GB and 4M files.
How is the standard deviation correlated to the block size and why?
If we now repeat such an experiment for many more (thousands) of hard disks, **which kind of distribution** do you get when you do a histogram of the used space of all hard disks?

4 Rest APIs & Block Chain Technology

Under https://pansophy.app:8443 you find a (hopefully running) Crypto Coin Server exporting a simple Block Chain.

You can open this URL in any web browser to see the current Block Chain status and the account information. At the time of writing the initial birth account of the Block Chain contained 1M coins (“genesis” : 1000000) :


```
{
    "accounts": {
        "genesis": 1000000
    },
    "chain": [
        {
            "coins": 1000000,
            "debit": "genesis",
            "index": 0,
            "my_hash":
"ac78160ca28ba2de7290551b66ff6e3f87e971f5bb068003306073e9d53f927a"
,
            "prev_hash": "0",
            "team": "genesis",
            "timestamp": 1622463916.974859
        }
    ],
    "current_hash":
"ac78160ca28ba2de7290551b66ff6e3f87e971f5bb068003306073e9d53f927a"
,
    "requests": {}
}
```


The REST responses are given in JSON format. Our REST API uses secure HTTP protocol and it is based on two HTTP methods:

GET
POST

GET requests are used, to retrieve any kind of information, POST requests are used to change state in the server.

The task is to implement a client and use a simple REST API to submit transactions to the Block Chain. Your goal is to book coins from other people's accounts to your own account.

The server implements a **Proof Of Time** algorithm. To add a transaction to move coins to your account, you have to submit a **merit** request and you have to let time pass before you can send a **claim** request to execute your transaction on the Block Chain. If you claim your transaction too fast after a merit request, your request is discarded. The server enforces a **Proof Of Time** of a minimum of 10 seconds!

The REST API documentation is given here:

I "GET" **Show accounts and Block Chain**

URL: https://pansophy.app:8443
METHOD: GET */**
RESPONSE: JSON
RESPONSE CODE: 200 OK
OUT FORMAT: (see example above)

* as mentioned if you past the URL into a web browser, your web browser executes a GET request and displays the response in your browser window

** you can execute this command using *curl* in your command line shell. If you don't have the curl command, you can install it using *sudo apt install curl*

To run a GET request with curl you can just do:

curl -k https://pansophy.app:8443

(the -k let's curl skip the certificate verification)

II “POST” **Initiate a transaction (merit)**

URL: https://pansophy.app:8443
METHOD: POST
DATA: JSON
RESPONSE CODE: 200 OK
DATA FORMAT:

```
{
   “operation”: “merit”,
   “team”: “Andreas J. Peters”,
   “coin”: 100,
   “steal from”: “genesis”
}
```

To call the merit function, you **POST** a JSON document with the format indicated above. Replace `team` with your team name, coin is the number of coins you want to transfer from account `steal from` (which has to exist!)

If you call the URL now (step I) with a GET request, you will see that an request has been added and it is visible in the JSON response:

```
...
"requests": {
        "Andreas J. Peters": {
            "coin": 100,
            "steal": "genesis",
            "team": "Andreas J. Peters",
            "timestamp": 1622464378.949318
        }
    }
```

If you submit several requests for a given team, they overwrite always the previous request - you can have only one request pending.

III “POST” **Conclude a transaction (claim)**

URL: https://pansophy.app:8443
METHOD: POST
DATA: JSON
RESPONSE CODE: 200 OK
DATA FORMAT:

```
{
   “operation”: “claim”,
        “team”: “Andreas J. Peters”,
     }
```

To claim your previously added merit request, you have to POST another JSON document as explained above respecting the **Proof of Time** requirement.

If you successfully claimed a transaction, you will see that coins have been booked on your team name account and the transaction is added to the Block Chain:


```
{
    "accounts": {
        "Andreas J. Peters": 1000,
        "genesis": 999000
    },
    "chain": [
        {
            "coins": 1000000,
            "debit": "genesis",
            "index": 0,
            "my_hash":
"58f3f485104a26b61060837a2087bbb5a14ba709e380dbaed452ed9e9cb39427",
            "prev_hash": "0",
            "team": "genesis",
            "timestamp": 1622479658.282847
        },
        {
            "coins": 1000,
            "debit": "genesis",
            "index": 1,
            "my_hash":
"baa43458ac89290b22c8e2e3fa0689265d47e45acb8e2980227f74eab01e23ac",
            "prev_hash":
"58f3f485104a26b61060837a2087bbb5a14ba709e380dbaed452ed9e9cb39427",
            "team": "Andreas J. Peters",
            "timestamp": 1622533027.813872
        }
    ],
    "current_hash":
"baa43458ac89290b22c8e2e3fa0689265d47e45acb8e2980227f74eab01e23ac",
    "requests": {}
```

4.1.1 Use the REST API and the `curl` command to **transfer coins** of the genesis or any other account on your own team account. You can use the `-d` option to POST a document. You have to indicate in your request, that the content type of the document is JSON. To do this you can add an HTTP header for this command

```
curl … -H"Content-Type: application/json" …
```

*If you prefer, you can use a Python program, doing the same HTTPS requests respecting **Proof of Time.** If you want to have some more fun, you can also load the current state into your Python script using GET requests and programatically steal from accounts which are reported. Be aware, that you can never steal the last coin of an account and if at the time of a claim there are not enough coins left on an account, your transaction is discarded.*

To you will have to add at least one successful transaction to the Block Chain.

4.1.2 What is the maximum number of transactions one given team can add to the Block Chain in one day?

4.2 The server has a function to compute a hash of a block in the Block Chain:

```
def calculate_hash(self):
    block_of_string = "{}{}{}{}{}".format( self.index,
                                           self.team,
                                           self.prev_hash,
                                           self.coins,
                                           self.timestamp)
    self.my_hash =
hashlib.sha256(block_of_string.encode()).hexdigest()
    return self.my_hash
```

4.2.1 Explain what this function does and why is this 'the key' for Block Chain technology?

4.2.2 If you have the knowledge of the hash function, **how can you validate** the contents of the Block Chain you received using a GET request to make sure, nobody has tampered with it? You don't need to implement

it! Explain the algorithm to validate a Block Chain!

4.2.3 **Why** might the **GET** REST API run into scalability problems? Express the scalability behaviour of execution times of GET and POST requests in **Big O notation** in relation to the number of transactions recorded in the Block Chain! Draw execution time vs transactions for GET and POST requests.

4.2.4 If the Crypto server goes down, the way it is implemented it loses the current account balances. **How can the server recompute the account balances** after a restart from the saved Block Chain?

4.2.5 **What are the advantages** of using a REST API and JSON in a client-server architecture? **What are possible disadvantages**?