UNIVERSITAT
ROVIRA i VIRGILI

Safe Distributed System Architectures

Assignment 3

Student:

Giorgio Rossi

January 21, 2021

Contents

1 Introduction

1.1 Project Repository

I have collected all the material related to this laboratory, scripts and results, on a Github repo [1], available at `jeorjebot/lithops-matrix-multiplication`.

The multiplication is coded inside the `matrix_multiplication.py` python script. In the `tests` directory I have provided some shell scripts that I used to compute the results, that are stored in the `data` directory, and showed on this report thanks to the Ipython notebook in the `notebooks` directory. The `reports` directory contains this report and all the images.

1.2 Libraries

This project used `Numpy` [2] to exploit the useful `ndarray` data structure shipped with this library, that is convenient for storing matrix. I used `Lithops` [3] as suggested, to be able to parallelise my code in local and ship my code unmodified in a serverless computing platform. Furthermore I used the `Phe` [4] library to compute the encryption with the Pailler cryptosystem [5].

1.3 Usage and requirements

The environment require `Lithops`, `Phe` and `Numpy` libraries, that you can easily install with the `requirements.txt` file provided:

```
python3 -m pip install -r requirements.txt
```

To run the code, that now is in `serverless` mode within the IBM Cloud, you have to specify the following parameters:

```
mode A_rows A_cols B_rows B_cols block_size
```

A simple example with two matrix, 10x20 and 20x10 and a size for row-blocks and column-blocks of 2, in plain and encrypted mode:

```
python3 matrix_multiplication.py plain 10 20 20 10 2
python3 matrix_multiplication.py encrypted 10 20 20 10 2
```

By default the multiplication is performed in `serverless` mode, but you can switch to `localhost` mode commenting line 126.

2 Implementation

The general architecture implemented consists of two phases: the **creation** of the matrices in the object storage, and the subsequent **multiplication**, computed with the fetching of the corresponding inputs, a row-block of the first matrix and a column-block of the second matrix, and then the saving of the

result submatrix in the object storage.

The creation of the matrices consist in saving within the storage a set of row-blocks, in the case of the first matrix, and a set of column-blocks, in the case of the second matrix, to make it easier the consequent matrix multiplication phase.

2.1 Plain Matrix Multiplication

The plain matrix multiplication process is the same explained above, where the first and the second matrices are in plain text.

2.2 Encrypted Matrix Multiplication

The encrypted matrix multiplication is similar to the plain matrix multiplication, with only a difference: the second matrix is **homomorphically encrypted** with the Pailler cryptosystem, that allows the multiplication with a plain matrix.

I have performed the encryption before the invocation of the serverless function that write the row-block / column-block in the object storage. This affects a lot the time execution since the encryption is very slow. In order to speed up this phase, a trick can be doing the encryption in the serverless function, in order to parallelise this task. I showed in the **Results** section that this idea is faster (approx 2x) compared to the local encryption, but it's not **secure**, because in this way the encryption is done in the cloud, allowing the honest-but-curious adversary to learn about our encrypted matrix.

3 Results

The experiments were carried out on a MacBook Pro (2014), with a 2,8 GHz Intel Core i5 dual-core and 8GB of RAM. The cloud chosen was the **IBM Cloud**.

For all the experiments the scale was composed of the following values: 1x, 2x, 5x, and where possible, 10x.

3.1 Increasing matrix sizes (local)

In this experiment the sizes of the matrix were 20x20, and the block size was fixed at 4. I have experimented increasing the number of rows in the first matrix, and the number of columns in the second matrix, i.e. an increase in the number of row-blocks in the first matrix and column-blocks in the second matrix. The consequence is an increase a workers deputed to manage that blocks. So the plot in Figure 1 was expected, due to the exponential numbers of workers created, and the bottleneck of the fixed number of cores available on my local machine.

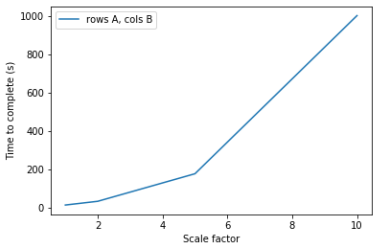

Figure 1: Increase in the number of **rows of the first matrix** and **columns of the second matrix**.

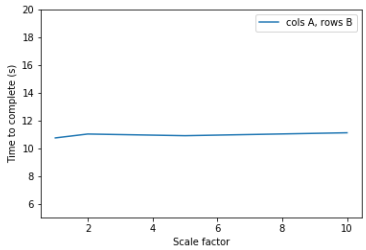

Figure 2: Increase in the number of **columns of the first matrix** and **rows of the second matrix**.

Increasing instead the number of columns in the first matrix, and the number of rows in the second matrix, which has no consequence on the number of workers but in the size of matrix provided in input to the serverless functions. The plot in Figure 2 is in line with this observation: the time to complete is not affected.

3.2 Increasing matrix sizes (cloud)

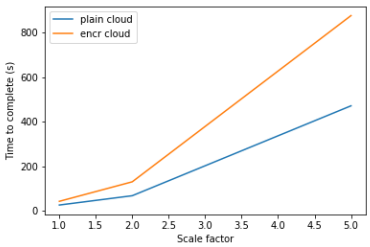

Figure 3: not scaled

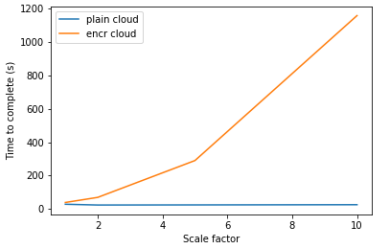

Figure 4: scaled

In this experiment I have increased first the sizes of the matrix, without increasing the block size parameter, in order to conseguently increase the number of workers. As expected I have obtained two exponential functions, that can be seen in Figure 3: of course the encrypted approach which have a steeper curve. I haven't plotted the 10x point because some error occured because of **Timeout** of the serverless function.

Then I have increased also the block size parameter with the same scale factor, and as expected the plain approach scale well, while the encrypted one is slow, as it can be seen in Figure 4, because of some bottleneck explained in the En-

crypted Matrix Multiplication section and below on the comparison of the two approaches.

This experiment was carried out on two 10x10 matrix, with scaling factors 1x, 2x, 5x, 10x and a block size parameter of 2. I have provided attached the lithops configuration: I used the **IBM Cloud** thanks to the Academic Initive of IBM.

3.3 Increasing block size parameter

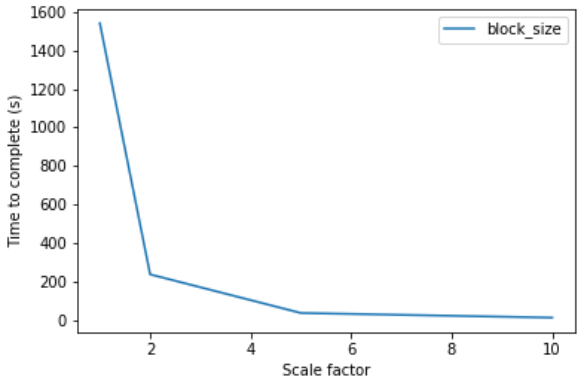

Figure 5: Scaling block size parameter on a multiplication of two 100x100 matrix

In this experiment I have gradually increased the block size parameter in order to group more rows in a row-block and columns in a column-block, in order to compute the result with **less workers**.

As shown in Figure 5, the computation speed decrease exponentially at the increase of that parameter, as expected. The sizes of the matrix were maintained fixed at (100x100) both, while the block size was scaled.

3.4 Plain vs Encrypted approach

In this experiment the matrix sizes and the block size parameter increase with the same scaling factor. The plain matrix multiplication maintain a low time to compute, thanks to the balance of increasing the matrix sizes (that increase the number of workers needed) and the increase of the block size parameter (that decrease the number of workers needed).

The two line which represent the encripted matrix multiplication approach have the big overhead of the homomorphical encryption, that is very slow. As said before in the Encrypted Matrix Multiplication section, the slower approach, in

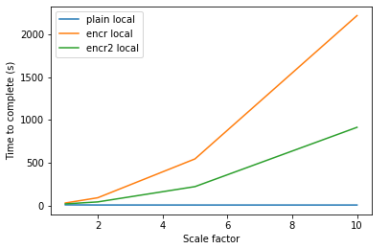

Figure 6: Scaling plain vs encrypted matrix multiplication on local

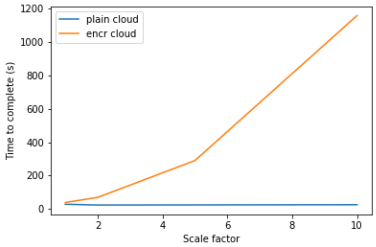

Figure 7: Scaling plain vs encrypted matrix multiplication on the cloud

orange, (the one present in the python final implementation) suffer from the bottleneck of computing the encryption in a not parallel way, without using a serverless function. Instead the second encrypted approach, in green, compute the encryption on a serverless function: is faster but less secure, because the data were sent ideally to the cloud and then encrypted. I have provided this second encrypted approach in the function `store_matrix_encrypted()` in my python script, without using that for the final implementation (kept as a heritage).

As I showed in Figure 6 and Figure 7, the results were quite similar. I can notice a better improvement in time with the encripted approach computed on the cloud that will needs further experiments.

The sizes were (10x10) for the first matrix and (10x10) for the second matrix. The scaling was applied on all the parameters, block size included.

References

[1] Giorgio Rossi. *Matrix multiplication, plain and encrypted, on the top of Lithops*. URL: https://github.com/jeorjebot/lithops-matrix-multiplication.

[2] Numpy Project. *Numpy, The fundamental package for scientific computing with Python*. URL: https://numpy.org.

[3] Josep Sampé et al. "Serverless Data Analytics in the IBM Cloud". In: Dec. 2018, pp. 1–8. DOI: 10.1145/3284028.3284029.

[4] CSIRO's Data61. *Python Paillier Library*. https://github.com/data61/python-paillier. 2013.

[5] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: vol. 5. May 1999, pp. 223–238. ISBN: 978-3-540-65889-4. DOI: 10.1007/3-540-48910-X_16.