КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технологій програмування

ЛАБОРАТОРНА РОБОТА

з дисципліни «Хмарні обчислення»

на тему: **Використання системи PARCS для паралельних обчислень**

Виконав:
студент групи ТТП-41
Єріс Євген

Київ – 2022

Зміст

Постановка задачі

Ознайомитися з використанням системи PARCS-Python шляхом написання програми для підрахунку норми Фробеніуса матриці та її тестування на різній кількості робочих віртуальних машин. Матриця складається з випадкових чисел, які генеруються при виконання програми.

Опис алгоритму

Норма Фробеніуса є векторною нормою, тому для її обчислення матриця m x n трактується як вектор розмірності mn. Для виконання паралельного обчислення матриця представляється у вигляді вектору й розбиваються на частини кількості, рівній числу воркерів. Отриманні результати з кожного воркера агрегуються в єдине число.

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$$

Код програми

Наступний уривок коду демонструє опис процедури введення даних із файлу, делегування роботи воркерам і підрахунку результату.

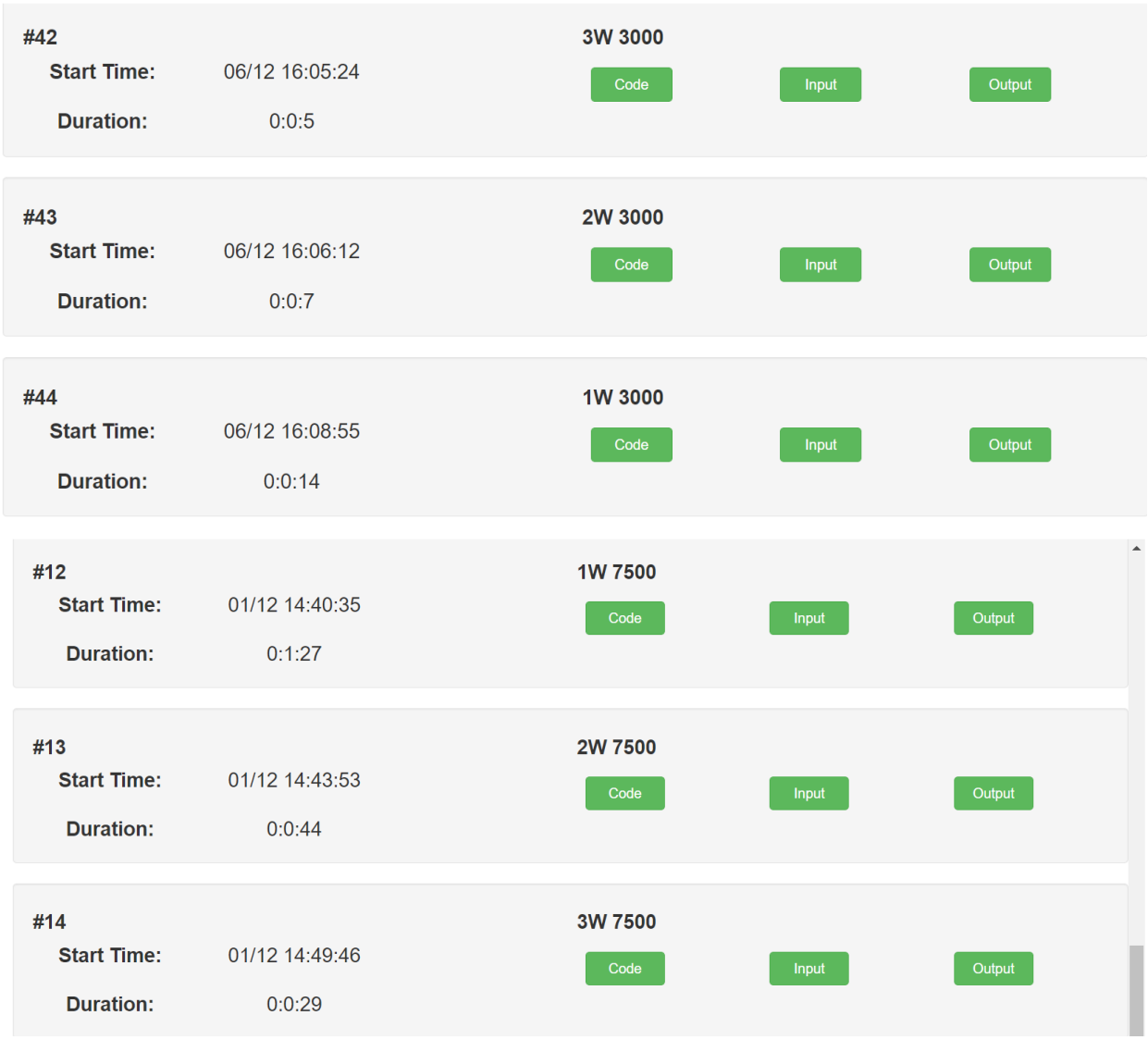
```
def solve(self):
        print("Job Started")
        print("Workers %d" % len(self.workers))

        data = self.read_input()
        m = int(data[0])
        n = int(data[1])
        step = m * n / len(self.workers)

        matrix = []
        mapped = []
        for i in xrange(0, len(self.workers)):
```

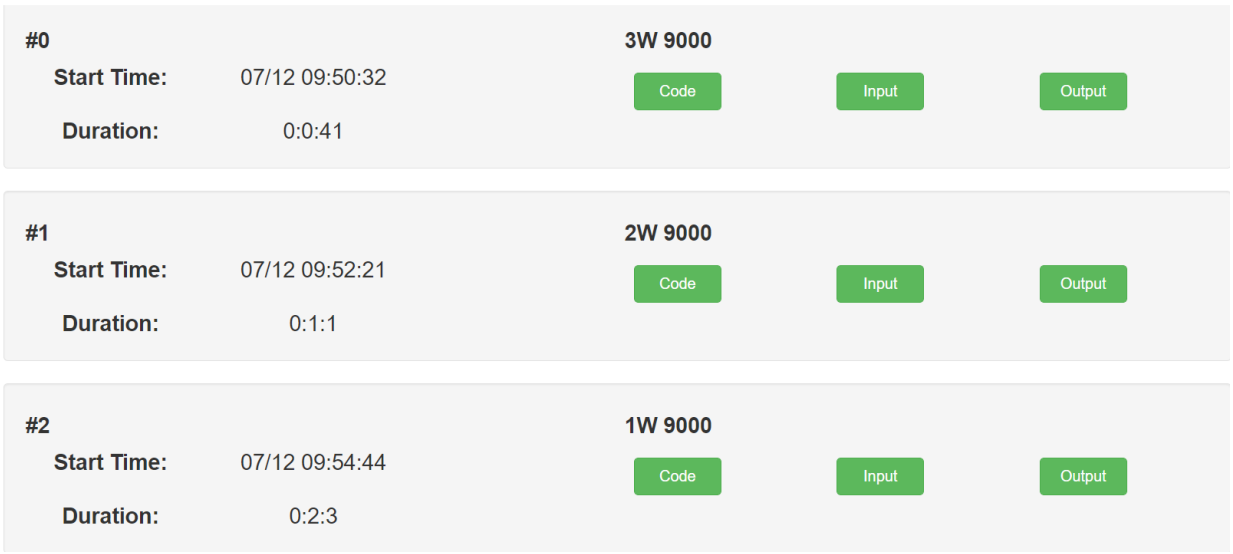
```
            result = self.workers[i].my_map(i * step, i * step + step)
            mapped.append(result)

        reduced, values = self.my_reduce(mapped)
        norm = math.sqrt(reduced)
        for x in values:
            self.append_output(x)
        self.append_output(norm)
        print("Job Finished")

    @staticmethod
    @expose
    def my_map(a, b):
        matrixPart = []
        result = 0

        for i in xrange(a, b):
             matrixPart.append(random.randint(0, 1000))

        for i in xrange(len(matrixPart)):
            result += matrixPart[i] * matrixPart[i]

        return result

    @staticmethod
    @expose
    def my_reduce(mapped):
        output = 0
        values = []

        for x in mapped:
            output += x.value
            values.append(x.value)

        return output, values

    def read_input(self):
        data = []
        f = open(self.input_file_name, 'r')
        data.append(f.readline().strip('\n').strip('\r'))
        data.append(f.readline().strip('\n').strip('\r'))
        f.close()
        return data

    def append_output(self, output):
```

```
        f = open(self.output_file_name, 'a')
        f.write(str(output) + '\n')
        f.close()
        print("output done")
```

Робота програми

Виконавши програму з використанням однієї, двох та трьох машин-воркерів для матриці розмірність 7500x7500, отримали наступні результати.





Виведення проміжних і кінцевих результатів:

1 робоча машина

```
18758565671716
4331115.98456
```

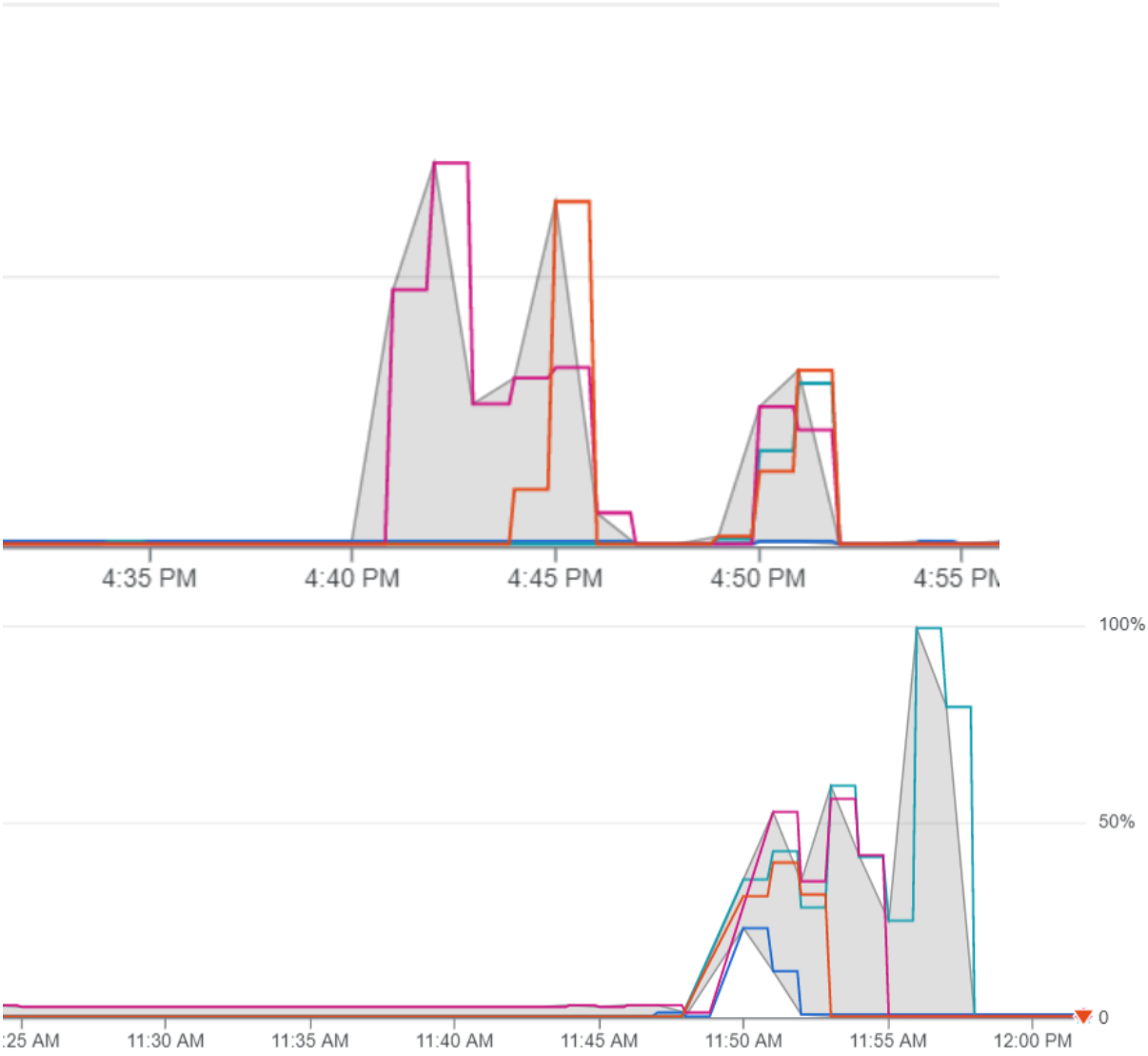
2 робочі машини

```
9378792141235
9379230945384
4331053.34608
```

3 робочі машини

```
6253969561778
6253026182300
6254698340330
4331477.125
```

Показник використання CPU під час виконання програми віртуальними машинами, наданий Google Cloud Monitoring:



Часові показники запусків програми:

	1 worker	2 workers	3 workers
3000x3000	0:0:14	0:0:7	0:0:5
7500x7500	0:1:27	0:0:44	0:0:29
9000x9000	0:0:41	0:1:1	0:2:3

Висновок

Отримані результати в ході виконання роботи свідчать про доцільність використання системи PARCS для виконання паралельних обчислень на декількох віртуальних машинах.

Використавши необхідний образ із Container Registry, за допомогою платформи Google Cloud були створені віртуальні машини, сконфігуровані для роботи з системою PARCS-Python. Для роботи з ними був використаний графічний інтерфейс master-машини, який дозволяє динамічно змінювати кількість worker-машин.

Написання лабораторної роботи дозволило отримати практичний досвід роботи з системою PARCS і покращити свої вміння роботи із хмарним сервісом Google Cloud.