Звіт до лабораторної роботи

з дисципліни "Хмарні обчислення"

на тему:

"Факторизація чисел алгоритмом Ро-Поларда з використанням технологій PARCS"

Виконала студентка групи ТТП-42

Формакідов Діна

Задача:

Реалізувати виконання алгоритму факторизації чисел алгоритмом Ро-Поларда з використання системи PARCS для мови програмування Python. Програма має приймати число і повертати добуток двох чисел з найбільшим простим множником. Проаналізувати прискорення, забезпечене використанням worker-ів та швидкість виконання програми в залежності від кількості.

Опис алгоритму:

Отже, хочемо факторизувати число n. Припустимо, що $n = pq$ и $p \approx q$. . Зрозуміло, що складнішого випадку, напевно, немає. Алгоритм ітеративно шукає найменший дільник і таким чином зводить завдання як мінімум удвічі менше.

Візьмемо довільну «досить випадкову» з погляду теорії чисел функцію. Наприклад, f(x)=(x+1)2modn.

Граф, у якому з кожної вершини є єдине ребро x→f(x), називається функціональним. Якщо в ньому намалювати «траєкторію» довільного елемента — якийсь шлях, що перетворюється на цикл, то вийде щось схоже на букву ρ (ро). Алгоритм тому так і названий.

Оригінальна версія

Розглядається послідовність цілих чиселx_n, така що$x_0 = 2$і$x_{i+1} = (x_i^2 - 1) \pmod N$, де$N$ - Число, яке потрібно факторизувати . Оригінальний алгоритм виглядає наступним чином :

1. Обчислюються трійки чисел

   $(x_i, x_{2i}, Q_i), i = 1, 2, \ldots,$ де$Q_i \equiv \prod_{j=1}^{i} (x_{2j} - x_j) \pmod N.$

   Причому кожна така трійка виходить із попередньої.

2. Щоразу, коли числоiкратно числуm(Скажімо,$m = 100$), обчислюється найбільший спільний дільник $d_i = \mathrm{GCD}(Q_i, N)$будь-яким відомим методом.
3. Якщо$1 < d_i < N$, то часткове розкладання числаNзнайдено, причому$N = d_i \times (N/d_i)$.

   Знайдений дільникd_iможе бути складеним, тому його також необхідно факторизувати. Якщо числоN/d_i складне, то продовжуємо алгоритм із модулем$N' = N/d_i$.

4. Обчислення повторюютьсяSразів. Якщо при цьому число не було до кінця факторизовано, вибирається, наприклад, інше початкове числоx_0.

Сучасна версія

НехайNскладове ціле позитивне число, яке потрібно розкласти на множники. Алгоритм виглядає наступним чином :

1. Випадковим чином вибирається невелика кількістьx_0і будується послідовність$\{x_n\}, n = 0, 1, 2, \ldots,$ визначаючи кожне наступне як$x_{n+1} = F(x_n) \pmod N.$
2. Одночасно на кожному *i* -му кроці обчислюється$d = \mathrm{GCD}(N, |x_i - x_j|)$для будь-яких$i, j$таких, що$j < i$ наприклад,$i = 2j$.
3. Якщо$d > 1$, то обчислення закінчується, і знайдене на попередньому етапі числоdє дільникомN. ЯкщоN/dне є простим числом, то процедуру пошуку дільників продовжується, взявши якNчисло$N' = N/d$.

На практиці функція$F(x)$вибирається не надто складною для обчислення (але водночас не лінійним багаточленом), за умови того, що вона не повинна породжувати однозначне взаємно відображення. Зазвичай як$F(x)$вибираються функції$F(x) = x^2 \pm 1 \pmod N$або$F(x) = x^2 \pm a \pmod N$. Однак функції$x^2 - 2$іx^2не підходять .

Якщо відомо, що для дільникаp числа Nсправедливо$p \equiv 1 \pmod k$при деякому$k > 2$, то має сенс використовувати $F(x) = x^k + b$.

Істотним недоліком алгоритму в такій реалізації є необхідність зберігати велику кількість попередніх значеньx_j.

Реалізація

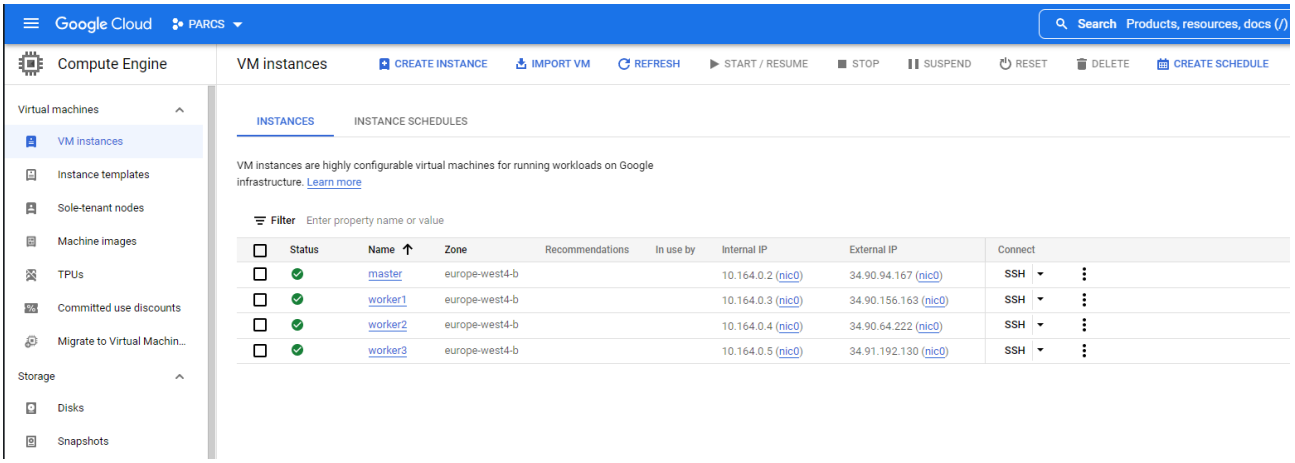
Нехай у нас є n ≥ 1 воркерів. Розбиваємо вхідний масив на n частин, які порівну розподіляємо між воркерами.

Тепер, реалізуємо обробку даних(факторизація чисел методом Ро-Поларда) для кожного воркера після того, як він приймає масив даних:

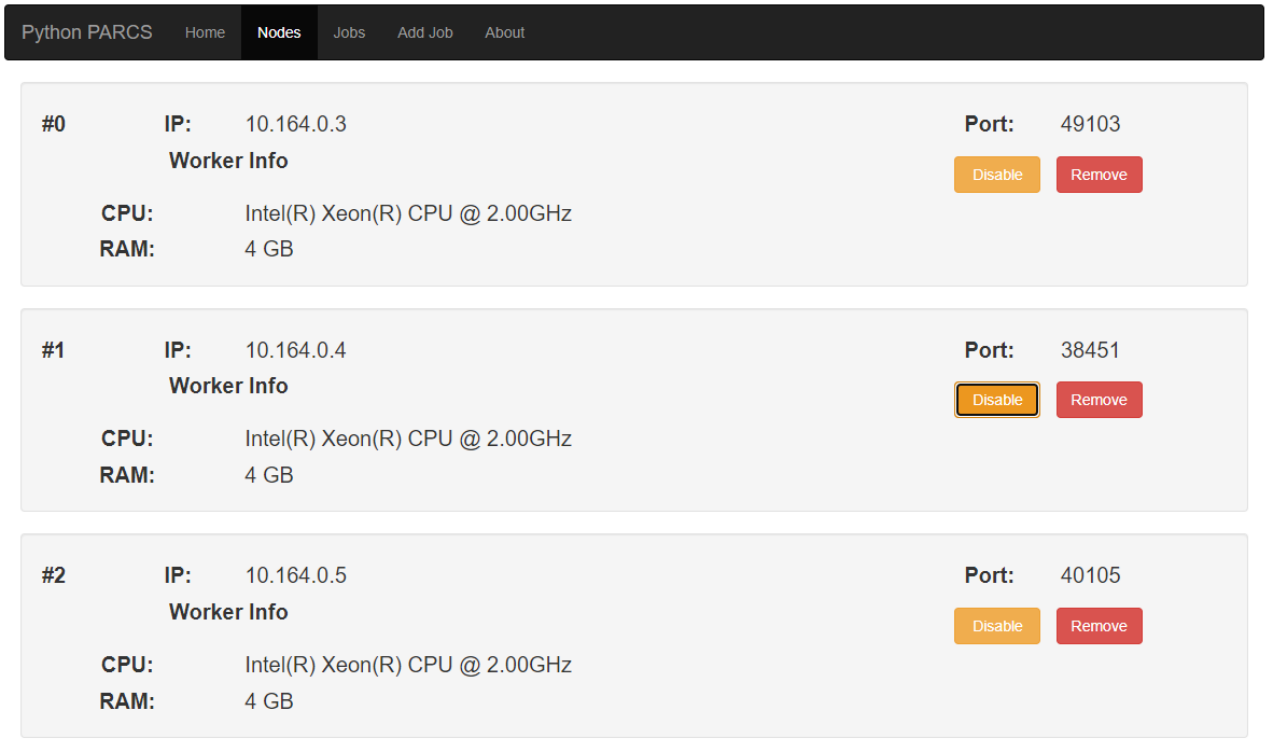
```
def domap(N, step):
    def gcd(x, y):
        while (y):
            x, y = y, x % y
        return x

    x = randint(1, N - 2)
    y = 1
    i = 0
    stage = 2
    while gcd(N, abs(x - y)) == 1:
        if i == stage:
            y = x
            stage *= 2
        x = (x * x + step) % N
        i += 1
    return gcd(N, abs(x - y))
```

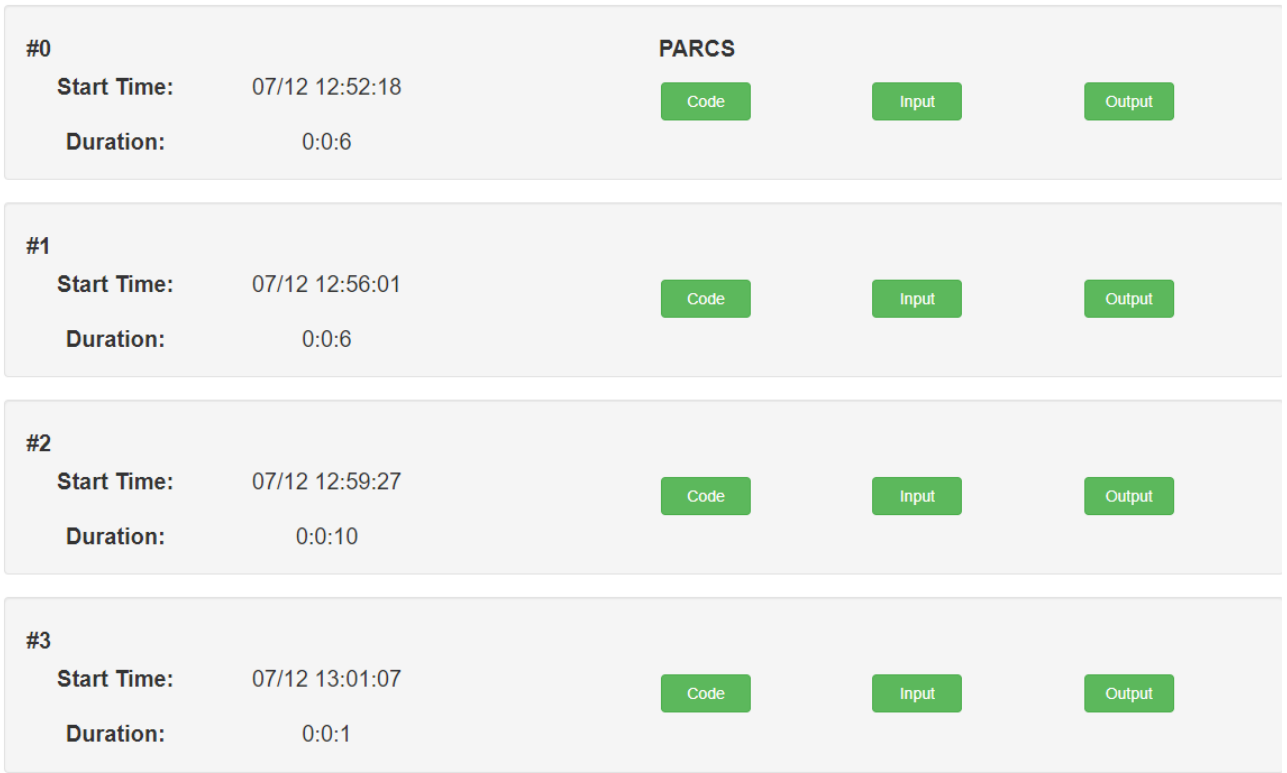
На платформі Google Cloud створюєму ВМ та воркерів:

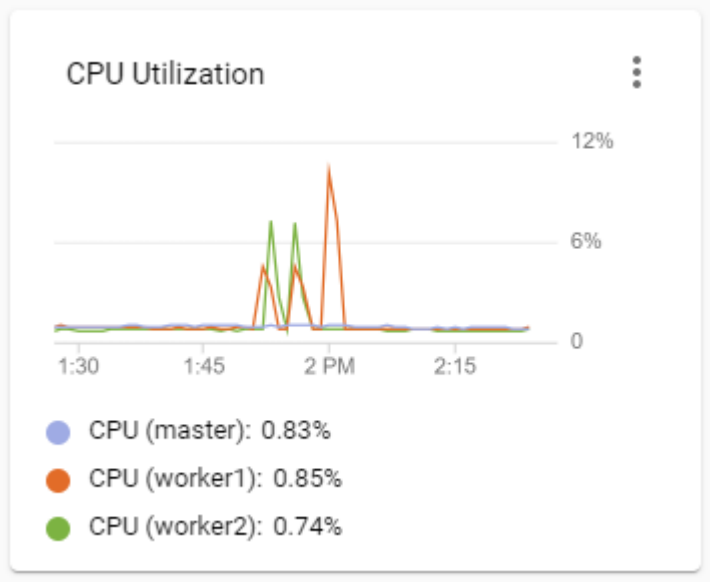


Заходимо на сайт, де бачимо список воркерів:



Додаємо Job та переглядаємо результати:





Результати спостереження:

Залежність часу виконання від найбільшого простого множника числа.

	10000000019	2550183799	193707721
1 worker	0:0:10	0:0:6	0:0:1
3 workers	0:0:6	0:0:2	0:0:1
7 workers	0:0:5	0:0:2	0:0:1

Висновок:

При виконанні лабораторної роботи було досліджено систему PARCS. Було зрівняно час виконання програми, в залежності від кількості воркерів та величини числа. З'ясовано, що зі збільшенням найбільшого простого множника, час виконання зростає.

Збільшення кількості воркерів зменшує час виконання програми, але тут немає шаблонної залежності, оскільки на це впливає як і алгоритм, так і вхідні дані та їх величина. В моєму випадку, три та сім воркерів відпрацювали майже за однаковий час. Проте, якщо порівнювати з одним воркером, то різниця може бути більш суттєва.

Вихідний код на GitHub

https://github.com/DinaZavr16/parcs