Politechnika
Warszawska

## Cloud Computing

# Estimating Pi Value using Monte Carlo method with help of Apache Spark

Mateusz Kozłowski

# 1 Introduction

Modern computers keep getting stronger and stronger. They are reaching new peaks when it comes to teraflops counts. However, future may lay elsewhere. Cluster-computing frameworks, such as Apache Spark combined with cloud platforms like Amazon Web Services allow us to carry out time-consuming calculations faster, without the need to build supercomputer.

Apache Spark is a well established framework with a large fanbase and a huge amount of tutorials, established in 2012. Amazon Web Services offer quiet big sum of money for academic purposes and have some free trial instances. These factors led me to choose previously mentioned platforms.

# 2 Project Overview

I decided to come up with a simple code, that takes significant time to execute so as to easily compare times of execution on a local machine, on one instance with one core, on a cluster consisting of Master and two Slaves - also one core instances and finally on one instance of with two cores.

The code, that is attached to this report, computates Pi Value using Monte Carlo method. Idea states that if you choose a large enough number of random points which are inside the square of dimensions 1 by 1 the proportion between points that are exactly 1 unit or closer to point (0,0) and all points is exactly $\frac{\pi}{4}$.

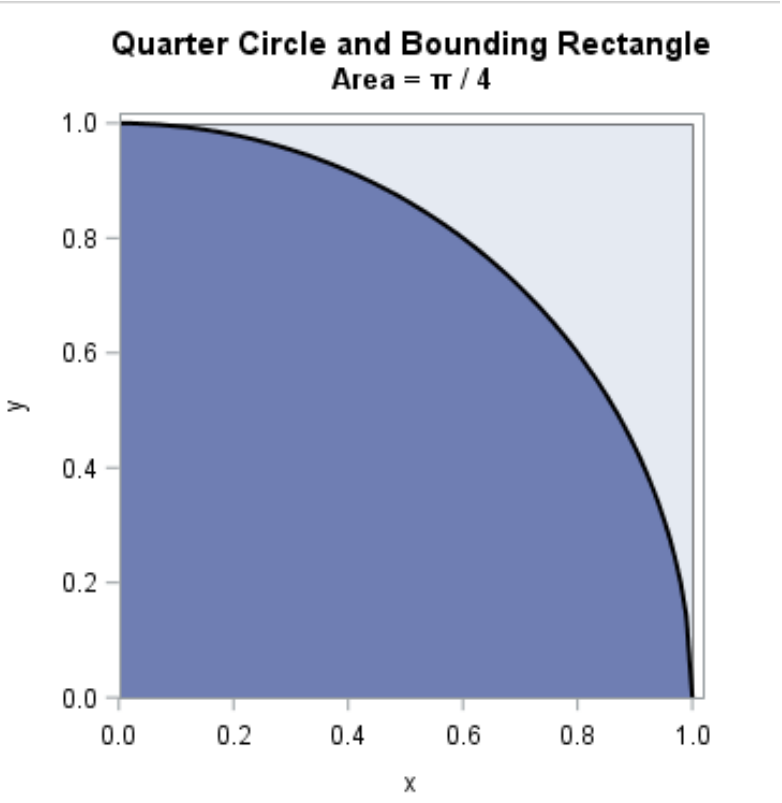



Figure 1: Idea of calculating Pi by Monte Carlo method

Code is simple, its main function is presented below.

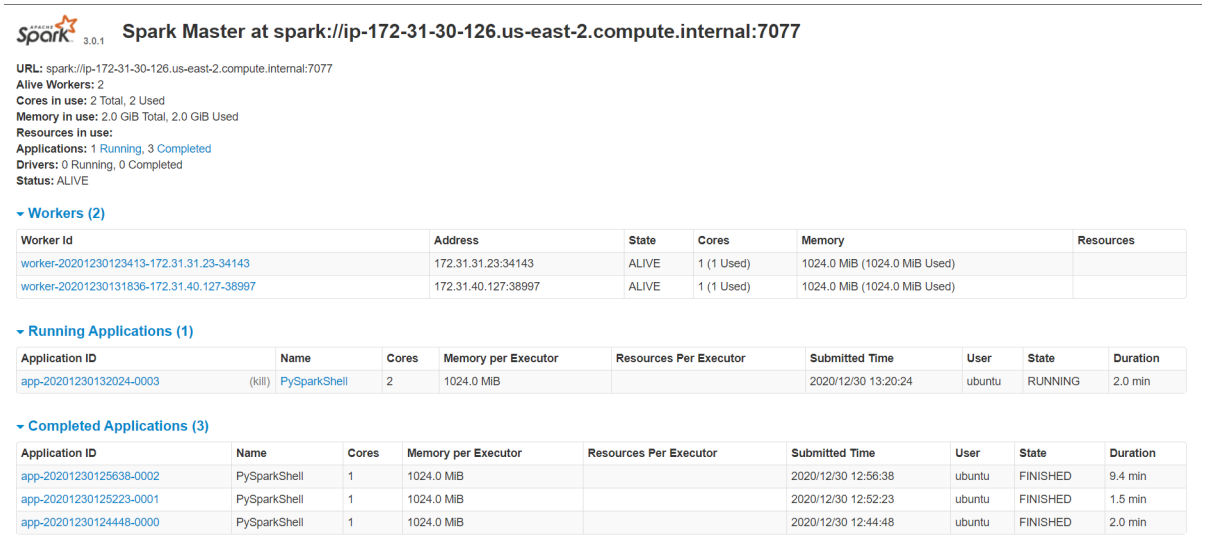
```
def monte_carlo(iterations):
    inside = 0
    for i in range(0, iterations):
        x = random.random()
        y = random.random()

        if math.sqrt(x*x + y*y) < 1.0:
            inside += 1
    return inside
```

Figure 2: Idea of Monte Carlo method converted to code

I prepared two versions of program - one working in single process and the other one working on parallel processes. Both were uploaded to my *Github* repository.

# 3 Steps

After writing the code comes the hard part. Now I had to establish, prepare and check the instances. Java, Scala and Spark was installed on every instance. Then three of instances were connected in previously mentioned cluster.



Spark 3.0.1 **Spark Master at spark://ip-172-31-30-126.us-east-2.compute.internal:7077**

**URL:** spark://ip-172-31-30-126.us-east-2.compute.internal:7077
**Alive Workers:** 2
**Cores in use:** 2 Total, 2 Used
**Memory in use:** 2.0 GiB Total, 2.0 GiB Used
**Resources in use:**
**Applications:** 1 Running, 3 Completed
**Drivers:** 0 Running, 0 Completed
**Status:** ALIVE

**Workers (2)**

| Worker Id | Address | State | Cores | Memory | Resources |
|---|---|---|---|---|---|
| worker-20201230123413-172.31.31.23-34143 | 172.31.31.23:34143 | ALIVE | 1 (1 Used) | 1024.0 MiB (1024.0 MiB Used) | |
| worker-20201230131836-172.31.40.127-38997 | 172.31.40.127:38997 | ALIVE | 1 (1 Used) | 1024.0 MiB (1024.0 MiB Used) | |

**Running Applications (1)**

| Application ID | Name | Cores | Memory per Executor | Resources Per Executor | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|---|
| app-20201230132024-0003 (kill) | PySparkShell | 2 | 1024.0 MiB | | 2020/12/30 13:20:24 | ubuntu | RUNNING | 2.0 min |

**Completed Applications (3)**

| Application ID | Name | Cores | Memory per Executor | Resources Per Executor | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|---|
| app-20201230125638-0002 | PySparkShell | 1 | 1024.0 MiB | | 2020/12/30 12:56:38 | ubuntu | FINISHED | 9.4 min |
| app-20201230125223-0001 | PySparkShell | 1 | 1024.0 MiB | | 2020/12/30 12:52:23 | ubuntu | FINISHED | 1.5 min |
| app-20201230124448-0000 | PySparkShell | 1 | 1024.0 MiB | | 2020/12/30 12:44:48 | ubuntu | FINISHED | 2.0 min |

Figure 3: Screenshot of Spark interface at the end of configuration

After assuring that everything is working I ran the code with the same number of iterations on four configurations

- Local Machine (i7-9750H)
- One instance (one core, 2.5GHz)
- Cluster (one Master, two slaves)
- One instance (two cores, 2.3GHz)

All calculations were carried out for the same number of iterations, that is 10 000 000. In the case of multiprocessing, number of processes multiplied by number of iterations per process was equal to 10 000 000. Every time mentioned in results is a mean average of 5 runs of the program.

I decided not to use *pyspark* library from python script, because I have encountered some problems downloading it on the instances. I launched my scripts directly from *PySpark* console, according to tutorial by Jay Sridhar on *dzone.com*

```
>>> os.system('python3 mp_off.py')
Number of iterations : 10000000
Estimation of PI : 3.1419432
Calculations took 3.5263173580169678
0
>>> os.system('python3 mp_on.py')
Number of processes : 10
Number of iterations per process: 1000000
Estimation of PI : 3.1408856
Calculations took 3.494095802307129s
0
```

Figure 4: Usage of and example output from program

# 4 Results

Local Machine :

$$SingleProcess : 7,945s$$

$$Multiprocessing : 1,661s$$

One instance (one core, 2.5GHz):

$$SingleProcess : 3,488s$$

$$Multiprocessing : 3,658s$$

Master + 2 x Slave :

$$SingleProcess : 3,403s$$

$$Multiprocessing : 3,468s$$

One Instance (two cores, 2.3GHz) :

$$SingleProcess : 3,478s$$

$$Multiprocessing : 1,702s$$

# 5 Conclusions

Local Machine was the slowest one to execute calculations in single process, however the fastest one to execute them in multiprocessing mode, due to processor consisting of six cores. Times in both cases of single instance, single process mode were similar. The multiprocessing was even a bit slower on a weaker instance due to some complications in code considering parallel processes. In my case using the cluster did not significantly cut on time needed to execute calculations. In my opinion the way I managed to force program to work in parallel process seems not to be working on clusters.

Cloud Computing is an powerful tool which allows to carry out calculations in acceptable time, even when on one is on budget. Even if, due to my incompetence, the cluster failed to beat the local machine, there is still a possibility of renting one, powerful instance only when you need it, which is certainly more cost-effective than buying a powerful PC.

# 6 Bibliography


- Correy Schafer, *Python Multiprocessing Tutorial: Run Code in Parallel Using the Multiprocessing Module*, https://www.youtube.com/watch?v=fKl2JWqrsot=1404s
- Oskar Bieńko *Deploying the Spark cluster including two nodes - master and slave.*, https://github.com/OskarBienko/Spark-cluster/blob/Spark-cluster/Deploying20the20Spark20cluster.md
- Jay Sridhar, https://dzone.com/articles/apache-spark-setting-up-a-cluster-on-aws
- *Tutorial: Getting started with Amazon EC2 Linux instances*, https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2GetStarted.html?r=1874
- *Connect to your Linux instance*, https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstances.html