Abstract

In this project we've been asked to code a program that execute the random walk algorithm using HTCondor in C.

Notes:

- The full code is attached in the zip file – RandomW.c .
- We've compiled the program with: gcc -o Random ./RandomW.c.
- The Submit file is attached in the zip file.
- We ran the histogram and data analysis with python (full code is attached).
- We used many condor commands, such as: condor submit, condor q, condor status, condor dagman, etc. (all the outputs/errors/logs/dags file are attached in the zip file).
- If any missing library in other computer, just run on command line: "pip install {name of library}".

Random Walk Algorithm

A random walk is a mathematical object, known as a stochastic or random process, that describes a path that consists of a succession of random steps on some mathematical space such as the integers. An elementary example of a random walk is the random walk on the integer number line, which starts at 0 and at each step moves +1 or -1 with equal probability.

In our Example, we used Random walk to simulate 10,000 particles which move as below:

- Up – +0.1 X axis.
- Down – (-0.1) X axis.
- Right - +0.1 Y axis.
- Left – (-0.1) Y axis.

We have a different probability to each direction as requested:

- Up: 0.15%
- Down: 0.15%
- Right: 0.65%
- Left: 0.05%

As illustrated by those examples, random walks have applications to many scientific fields including ecology, psychology, computer science, physics, chemistry, biology as well as economics. Random walks explain the observed behaviors of many processes in these fields and thus serve as a fundamental model for the recorded stochastic activity. As a more mathematical application, the value of pi can be approximated by the usage of random walk in the agent-based modeling environment.

Condor Submit

First, after coding and compiling the program, we created the submit file as below:

Universe = vanilla

executable = Random // Name of our executable

output = Random_$(Process).out

error = Random_$(Process).error

log = Random_$(Process).log

Arguments = $(Process)

Queue 50 // 50 as requested

```
Submitting job(s).................................................
50 job(s) submitted to cluster 1537.
hobbit3.ee.bgu.ac.il> condor_q


-- Schedd: hobbit3.ee.bgu.ac.il : <132.72.53.3:9618?... @ 01/05/22 12:28:17
OWNER  BATCH_NAME     SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
roieka CMD: Random   1/5  12:02     92      8     50    150 1535.40 ... 1537.49

58 jobs; 0 completed, 0 removed, 50 idle, 8 running, 0 held, 0 suspended
hobbit3.ee.bgu.ac.il> condor_q


-- Schedd: hobbit3.ee.bgu.ac.il : <132.72.53.3:9618?... @ 01/05/22 12:28:22
OWNER  BATCH_NAME     SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
roieka CMD: Random   1/5  12:02     92      _     58    150 1535.40 ... 1537.49

58 jobs; 0 completed, 0 removed, 58 idle, 0 running, 0 held, 0 suspended
```

Pic 1: Condor submit and condor q right after.

From Pic 1 we are are able to see the difference between the first condor q to the second time we executed condor q: in the first time we ran it: 8 tasks were running, and in the second time all of them finished.

Condor Status

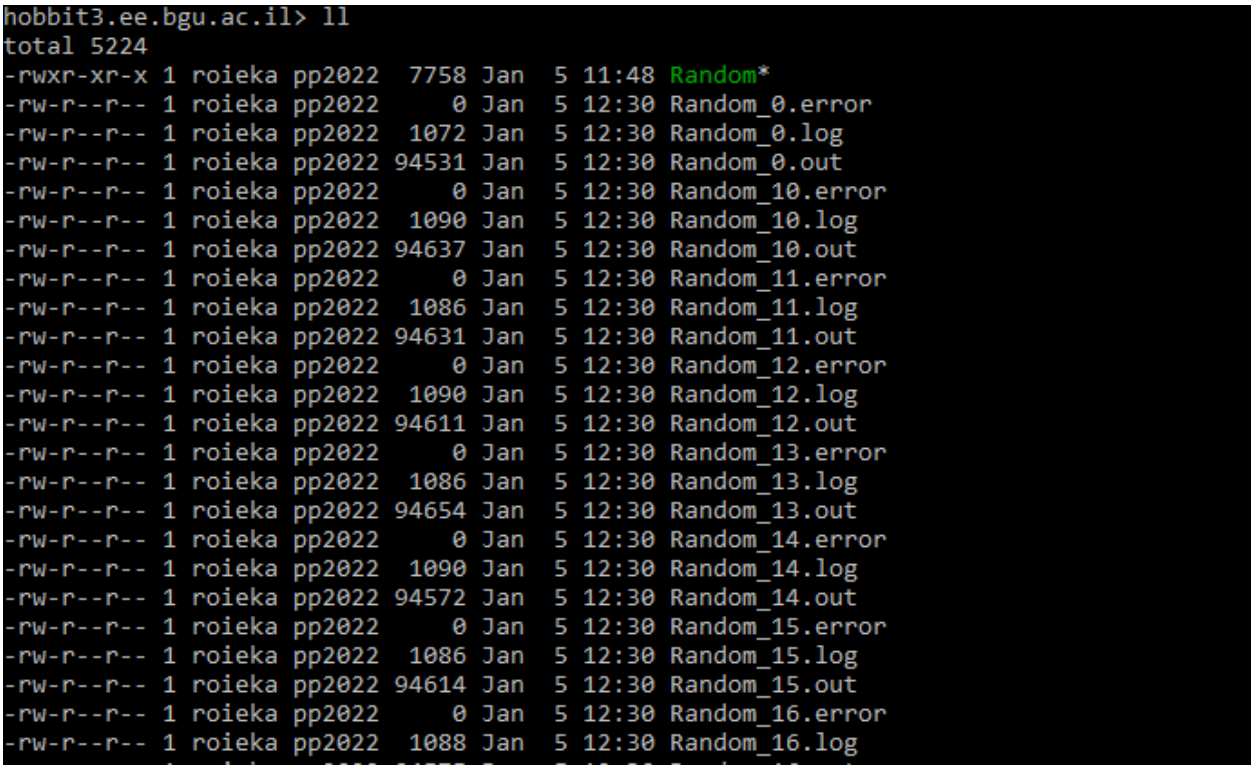
```
hobbit3.ee.bgu.ac.il> condor_status
Name                          OpSys      Arch   State     Activity LoadAv Mem   ActvtyTime

slot1@hobbit1.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  0+00:00:02
slot2@hobbit1.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  0+00:00:01
slot3@hobbit1.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.100 1942  0+00:00:02
slot4@hobbit1.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  0+00:00:02
slot5@hobbit1.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  0+00:00:02
slot6@hobbit1.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  0+00:00:02
slot7@hobbit1.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  0+00:00:02
slot8@hobbit1.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  0+00:00:02
slot1@hobbit2.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  3+11:16:18
slot2@hobbit2.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  3+11:16:22
slot3@hobbit2.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  3+11:16:23
slot4@hobbit2.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  3+11:14:26
slot5@hobbit2.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  4+23:49:33
slot6@hobbit2.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  4+23:49:34
slot7@hobbit2.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  4+23:49:34
slot8@hobbit2.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  4+23:49:27
slot1@hobbit3.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  0+00:03:42
slot2@hobbit3.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  0+00:03:42
slot3@hobbit3.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  0+00:03:42
slot4@hobbit3.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  0+00:03:42
slot5@hobbit3.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  0+00:03:42
slot6@hobbit3.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  0+00:03:42
slot7@hobbit3.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.010 1942  0+00:03:42
slot8@hobbit3.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.000 1942  0+00:03:42
slot1@hobbit4.ee.bgu.ac.il  LINUX      X86_64 Unclaimed Idle      0.050 1942  4+19:50:19
```

Pic 2: Output of Condor Status.

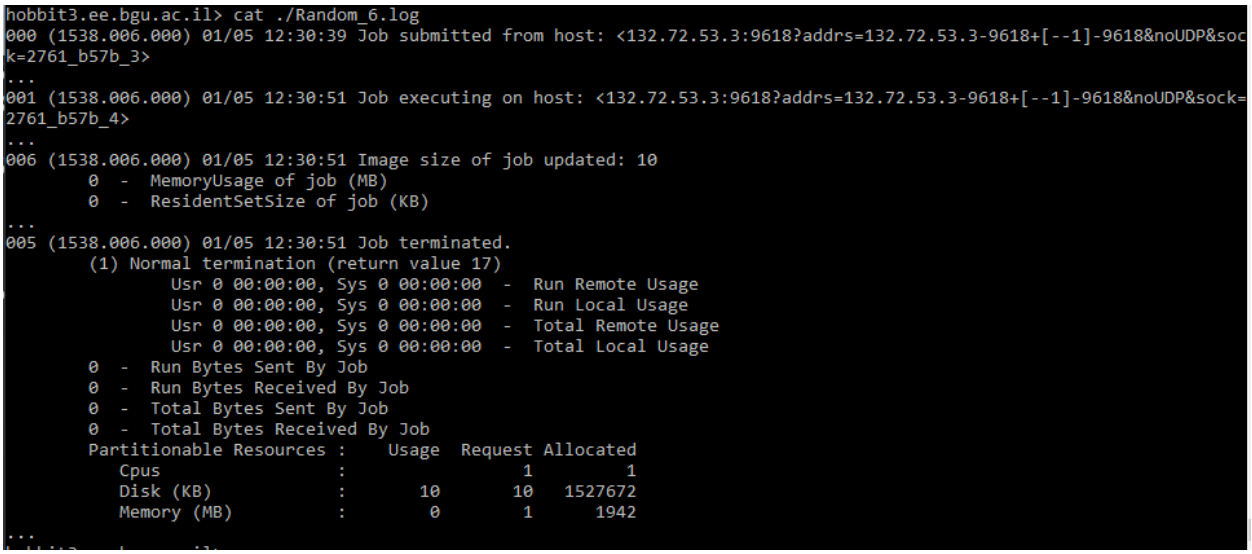
Files that have been created

Right after we ran condor status, we checked the if all the files have been created successfully.

What we wanted to see that all error files are empty, and output are correct.



Pic 3: output of the list in our directory.



Pic 4: Check one output file for example.

Creating Histogram

We created a program that takes all the output data and create a visualization of the case.

We used python and its regular libraries for data analysis such as: pandas, numpy.

We used seaborn library for great data visualization.

In [1]:
```
#pip install "Library" - if a library is missing
import matplotlib
import collections
from collections import Counter
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

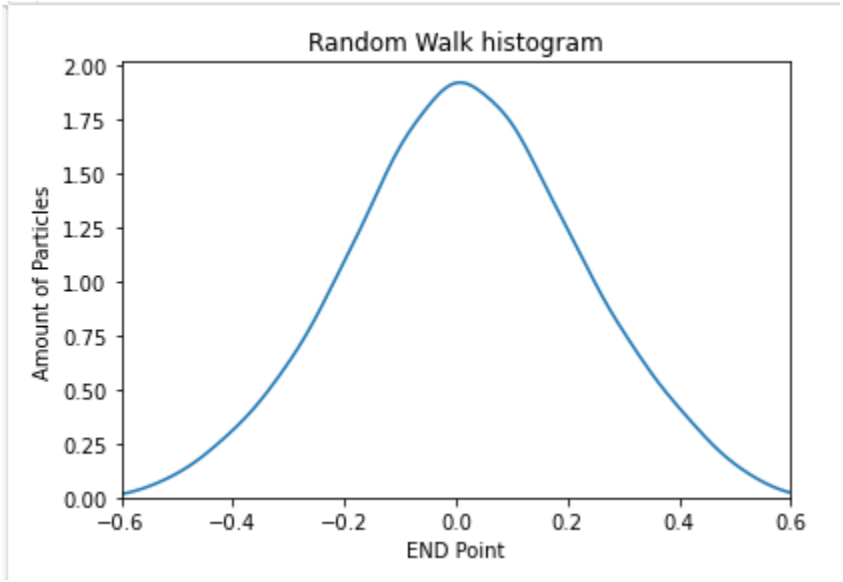
In [2]:
```
arr =[] # creating an empty array and insert to it the list of each output
for i in range(50):
    with open('./Random_'+str(i)+'.out') as output:
        current = list(output)
        for j in range(len(current)-3):
            arr.append(float(current[j+3]))
```

In [3]:
```
#Count of the number of appearances of each result
appear = Counter(arr)
appear
```

Out[3]:
```
Counter({0.2: 61480,
         0.1: 88863,
         0.0: 100397,
         -0.3: 29625,
         -0.1: 82958,
         0.3: 37066,
         -0.2: 53590,
         0.4: 20084,
         -0.4: 14668,
         0.5: 6510,
         -0.5: 4659})
```

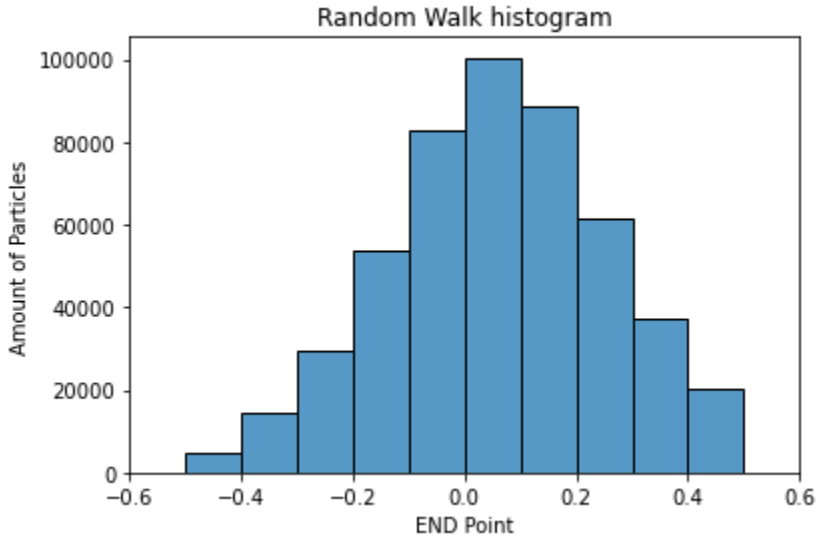
Pic 5: Importing libraries, creating array from all output files, creating a counter that show all the number of appearances of each result.

```
In [4]: sns.kdeplot(arr, bw_adjust=4)
        plt.xlim(-0.6,0.6)
        plt.title('Random Walk histogram')
        plt.ylabel('Amount of Particles')
        plt.xlabel('END Point')
        plt.show()
```



Pic 6 & 7: Input & output for continuous histogram.

```
In [5]: sns.histplot(arr,binwidth=0.1)
        plt.xlim(-0.6,0.6)
        plt.title('Random Walk histogram')
        plt.ylabel('Amount of Particles')
        plt.xlabel('END Point')
        plt.show()
```



Pic 8 & 9: Input & output for regular histogram.

Conclusion & Insights

In this assignment, we've deeply learned how to run independent variables with HTCondor.

We deeply find it helpful specially to run the same program with different variables.

HTCondor takes advantage of computing resources that would otherwise be wasted and puts them to good use. HTCondor streamlines the programer's tasks by allowing the submission of many jobs at the same time. In this way, tremendous amounts of computation can be done with very little intervention from the user. Moreover, HTCondor allows users to take advantage of idle machines that they would not otherwise have access to.

We will definitely try to use HTCondor for one of our Deep Learning project, where we want to run the same Neural Network with different parameters (batch size, epochs, learning rate, etc.).

2 Open Question on what we've learned so far on HTCondor:

***Right answers will be marked in YELLOW**

Question number 1:

How do we give numbering to each job in HTCondor?

1. Number of process (for example: 1,2,3, etc.).
2. Run a loop (pseudo code: for i in jobs: print(i)).
3. Cluster.Process (for example 3.1,4.2, etc.).
4. Alphabet order (for example a,b,c, etc.).

Question number 2:

When will Condor produce a checkpoint?

1. Periodically, if desired, for fault tolerance.
2. When the job is preempted, either by a higher priority job, or because the execution machine becomes busy.
3. When the user explicity runs a condor_checkpoint, condor_vacate, condor_off or condor_restart command.
4. All the answers are correct.