

# CPU ALLOCATION

CPU is one of the critical resources in a distributed processing methodology containing runtime requirement. This project involves reading the historical log files containing the information of CPU, runtime, and coming up with the number of CPUs that can be allocated to the upcoming tape out that needs to be done in a given time frame.

For this programming challenge log files of four layers LayerA, LayerB, LayerC, LayerD

- Each containing 3 log files with information of a particular sub layer's start time, finish time, and its CPU usage
- An excel sheet containing information about run time estimation of the upcoming layers has been provided.

As a part of this programming challenge, I have

- Analyzed each log file of 4 layers
- Extracted the required details from each log files
- Modified/Cleaned the extracted data
- Studied the relationship between CPU-runtime for each layer
- Computed the average CPU-runtime value of each layer
- Calculated the number of CPUs required for the upcoming layers based on the run time requirement provided for the new layers
- Updated the excel file with number of CPUs required for each new layer

### How does the code work?

This project has been implemented in python using PyCharm editor. The approach followed to calculate the number of CPUs for the new layers are mentioned in the steps below.

- Placed all the log files in a single folder and providing that path in the program for read operation.
- Iterated through all the log files based on the name of the file. For example, if the file name has the substring 'LayerA', it will loop through all the LayerA log files.
- Once all the LayerA log files are collected, the code reads the content of each log file of LayerA and manipulates its content.
- Since the log file has other information like Job ID, step details, exit status, which is not necessary for this project requirement, a function 'filter\_data' is written which takes the log file as an input. This function has a list of column values that we require to do the analysis (Logfile, Starting time, Finishing time, CPU Usage)

```

+-----+
| Logfile      : LayerA_0001.log |
| Job id       : 784059256      |
| Queueing time : Wed Dec 23 17:42:06 2020 |
| Starting time  : Wed Dec 23 17:42:08 2020 |
+-----+

```

Running at LayerA\_0001 to the end of step(s)

```

Step1 Completed
Step2 Completed
Step3 Completed

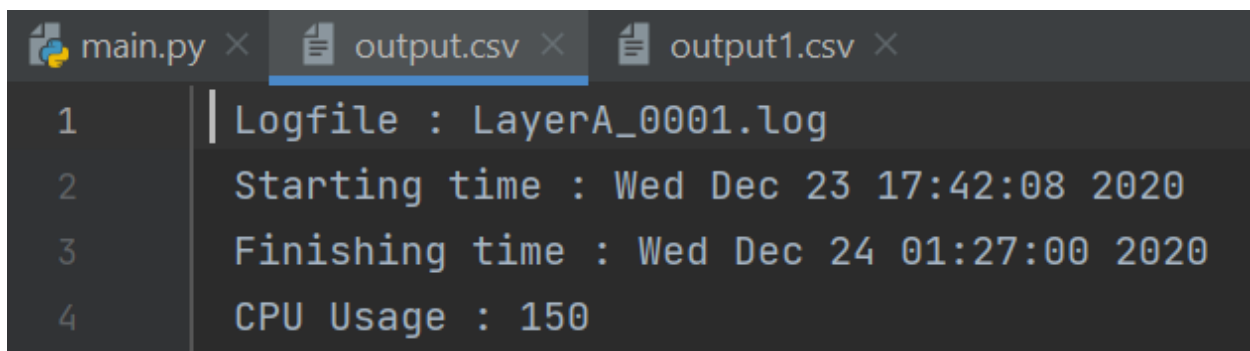
```

```

+-----+
| Exit Status   : 0              |
| Finishing time : Wed Dec 24 01:27:00 2020 |
| CPU Usage      : 150           |
+-----+

```

- This function reads each log file and returns all the lines of the file that contain the above strings in a list.
- When all the matched lines of a particular log file are returned in a list, it is further cleaned by eliminating the special characters (|) and unnecessary strings (\n) and spaces.
- The cleaned list is then written to the output.csv file.
- Now the data in output.csv for a particular log file looks something like below



```

1 | Logfile : LayerA_0001.log
2 | Starting time : Wed Dec 23 17:42:08 2020
3 | Finishing time : Wed Dec 24 01:27:00 2020
4 | CPU Usage : 150

```

- After the list is stored in the csv file, it is read line by line and then split the lines when finds the symbol (:). A dataframe is created with the key data from the value before (:) and value data from the value after (:) symbol.
- The dataframe now has columns and rows in the below format

Logfile	Starting Time	Finishing Time	CPU Usage
LayerA_001.log	Wed Dec 23 17:42:08 2020	Wed Dec 24 01:27:00 2020	150

```
C:\Users\deepu\PycharmProjects\CPUSchedulin
      Logfile      ...  CPU Usage
0  LayerA_0001.log  ...      150

[1 rows x 4 columns]
```

- To calculate the total run time (Finishing Time – Starting Time) for each log file, the StartTime and Finishing time columns are extracted from the dataframe and converted into datetime format (e.g., 12/23/2020 17:42:08 2020). The total time is then calculated with the formatted columns.
- The calculated total run time is rounded off and converted into number of hours.
- Now we have the total run time and CPU usage for each log file. These two values are appended to another output1.csv file for each log file. The relationship between CPU-runtime is observed to be linear. That is, as the number of CPUs increase, runtime becomes faster which means for the larger value of CPU, runtime value decreases (faster runtime).
- Now we have the 3 log file details of LayerA (LayerA\_001, LayerA\_002, LayerA\_003) at this point.
- All these values of 3 log files are then averaged to form a single average\_runtime and average\_cpu for the LayerA.
- These average values are sent to another function 'calculate\_cpu\_allocation' which takes these two values as an input and calculates based on the inverse proportion formula.
  - k is the [constant of proportionality](#).
  - y increases as x decreases.
  - y decreases as x increases.

Inverse Proportion Formula



$$y \propto \frac{1}{x}$$

$$y = k \left( \frac{1}{x} \right)$$

$$y = \frac{k}{x}, \text{ for a constant } k$$

- Finally, the excel file with the upcoming layer information is accessed and updated with the calculated CPU allocation value for the new LayerA
- Now we have CPU allocation value for the upcoming LayerA.
- The program then loops and follows the same procedure for the remaining LayerB, LayerC, and LayerD.

### **How long did you spend working on the problem? What did you find to be the most challenging part?**

I have taken a good 15-20 minutes of time to understand the question and log files. I have then taken another 10 minutes to calculate the CPU allocation values manually just to cross check after the program is written. I have then started working on programming on which I have spent around 3-4 hours. Another 1 hour has been spent on the documentation. The total time taken to understand, analyze, program, and document is around 5 to 5 and half hours. I enjoyed working on this project and found it to be not very tough to implement. If I must compare the phase that I found more challenging, I would say it is the manipulation or cleaning the data since that is the most critical part. Once the data is manipulated/cleaned, calculations are comparatively easy to perform.

### **What other factors do you think might affect the runtime other than CPU #? What other information would be helpful for better run-time prediction?**

Some of the factors that I think might affect the runtime other than CPU # are

- Algorithm that is chosen to build the program (Time Complexity)
- Operating system
- Input data or number of steps involved in a process to execute
- Multithreading yields better run time
- Architecture of the system
- Language that is used to build the program

All these above-mentioned factors I think might affect the runtime and if we have any additional information regarding these factors would be helpful in better run-time prediction. Since we are predicting values, machine learning techniques could also be considered and implemented to yield accurate results when working with large set of data.

### **How to execute the program?**

- Unzip the project folder
- Import the whole 'CPUScheduling' folder into the PyCharm editor or any other python editor
- Open the main.py

- Click on Run
- Now open the excel sheet to find the updated values

## References

Image from - [Inverse Proportion - Formula, Examples, Definition, Graph \(cuemath.com\)](https://www.cuemath.com/inverse-proportion-formula/)