taskscheduling

January 30, 2020

[1]:
```
import queue as Q
import numpy as np
import matplotlib.pyplot as plt
import random
from operator import itemgetter
%matplotlib inline
```

[2]:
```
random.seed(1234)
```

[3]:
```
def generate(clusters,no_task,arrival_together):
    test_case = []
    for i in range(no_task):
        clusterno = random.randint(1,clusters)
        workload = random.randint(1,25)
        if(arrival_together  == False):
            arrival_time = random.randint(1,10)
            completion_time = random.
 ↪randint(arrival_time+workload,arrival_time+workload+100)
        else:
            arrival_time = 0
            completion_time = 1000
        test_case.append((clusterno,arrival_time,workload,completion_time,i+1))
    return test_case
def generateworker(workerno):
    processors = []
    for i in range(workerno):
        processors.append(random.randint(11,21))
    return processors
```

[4]:
```
def sjf(cur_test,workers):
    task_cnt = len(cur_test)
    worker_cnt = len(workers)
    result = []
    ready_time = []
    cur_test.sort(key = itemgetter(2))
    cur_test.sort(key = itemgetter(0))
    cur_test.sort(key = itemgetter(1))
```

```
    for i in range(worker_cnt):
        ready_time.append(0)
    for x in cur_test:
        min_time = 100010
        for i in range(worker_cnt):
            estimated_time = x[2]/workers[i]+x[1]+max(0,ready_time[i]-x[1])
            if(estimated_time<min_time):
                min_time = estimated_time
                cur = i
        ready_time[cur] = min_time
        result.append((x[4],cur,min_time))
    return result
```

[5]:
```
def fcfs(cur_test,workers):
    task_cnt = len(cur_test)
    worker_cnt = len(workers)
    result = []
    ready_time = []
    cur_test.sort(key = itemgetter(0))
    cur_test.sort(key = itemgetter(1))
    for i in range(worker_cnt):
        ready_time.append(0)
    for x in cur_test:
        min_time = 100010
        for i in range(worker_cnt):
            estimated_time = x[2]/workers[i]+x[1]+max(0,ready_time[i]-x[1])
            if(estimated_time<min_time):
                min_time = estimated_time
                cur = i
        ready_time[cur] = min_time
        result.append((x[4],cur,min_time))
    return result
```

[6]:
```
def ljf(cur_test,workers):
    task_cnt = len(cur_test)
    worker_cnt = len(workers)
    result = []
    ready_time = []
    cur_test.sort(key = itemgetter(2),reverse = True)
    cur_test.sort(key = itemgetter(0))
    cur_test.sort(key = itemgetter(1))
    for i in range(worker_cnt):
        ready_time.append(0)
    for x in cur_test:
        min_time = 100010
        for i in range(worker_cnt):
            estimated_time = x[2]/workers[i]+x[1]+max(0,ready_time[i]-x[1])
```

```
            if(estimated_time<min_time):
                min_time = estimated_time
                cur = i
        ready_time[cur] = min_time
        result.append((x[4],cur,min_time))
    return result
```

[7]:
```
def edf(cur_test,workers):
    task_cnt = len(cur_test)
    worker_cnt = len(workers)
    result = []
    ready_time = []
    cur_test.sort(key = itemgetter(3))
    cur_test.sort(key = itemgetter(0))
    cur_test.sort(key = itemgetter(1))
    for i in range(worker_cnt):
        ready_time.append(0)
    for x in cur_test:
        min_time = 100010
        for i in range(worker_cnt):
            estimated_time = x[2]/workers[i]+x[1]+max(0,ready_time[i]-x[1])
            if(estimated_time<min_time):
                min_time = estimated_time
                cur = i
        ready_time[cur] = min_time
        result.append((x[4],cur,min_time))
    return result
```

[8]:
```
def random_assignment(cur_test,workers):
    task_cnt = len(cur_test)
    worker_cnt = len(workers)
    result = []
    ready_time = []
    random.shuffle(cur_test)
    cur_test.sort(key = itemgetter(0))
    cur_test.sort(key = itemgetter(1))
    for i in range(worker_cnt):
        ready_time.append(0)
    for x in cur_test:
        min_time = 100010
        for i in range(worker_cnt):
            estimated_time = x[2]/workers[i]+x[1]+max(0,ready_time[i]-x[1])
            if(estimated_time<min_time):
                min_time = estimated_time
                cur = i
        ready_time[cur] = min_time
        result.append((x[4],cur,min_time))
```

```
    return result
```

[9]:
```
def analyze(x):
    reso = {}
    max_time = 0
    for xx in x:
        max_time = max(0,xx[2])
        if xx[1] in reso:
            reso[xx[1]].append(xx[0])
        else:
            reso[xx[1]]=[]
            reso[xx[1]].append(xx[0])
    return max_time
```

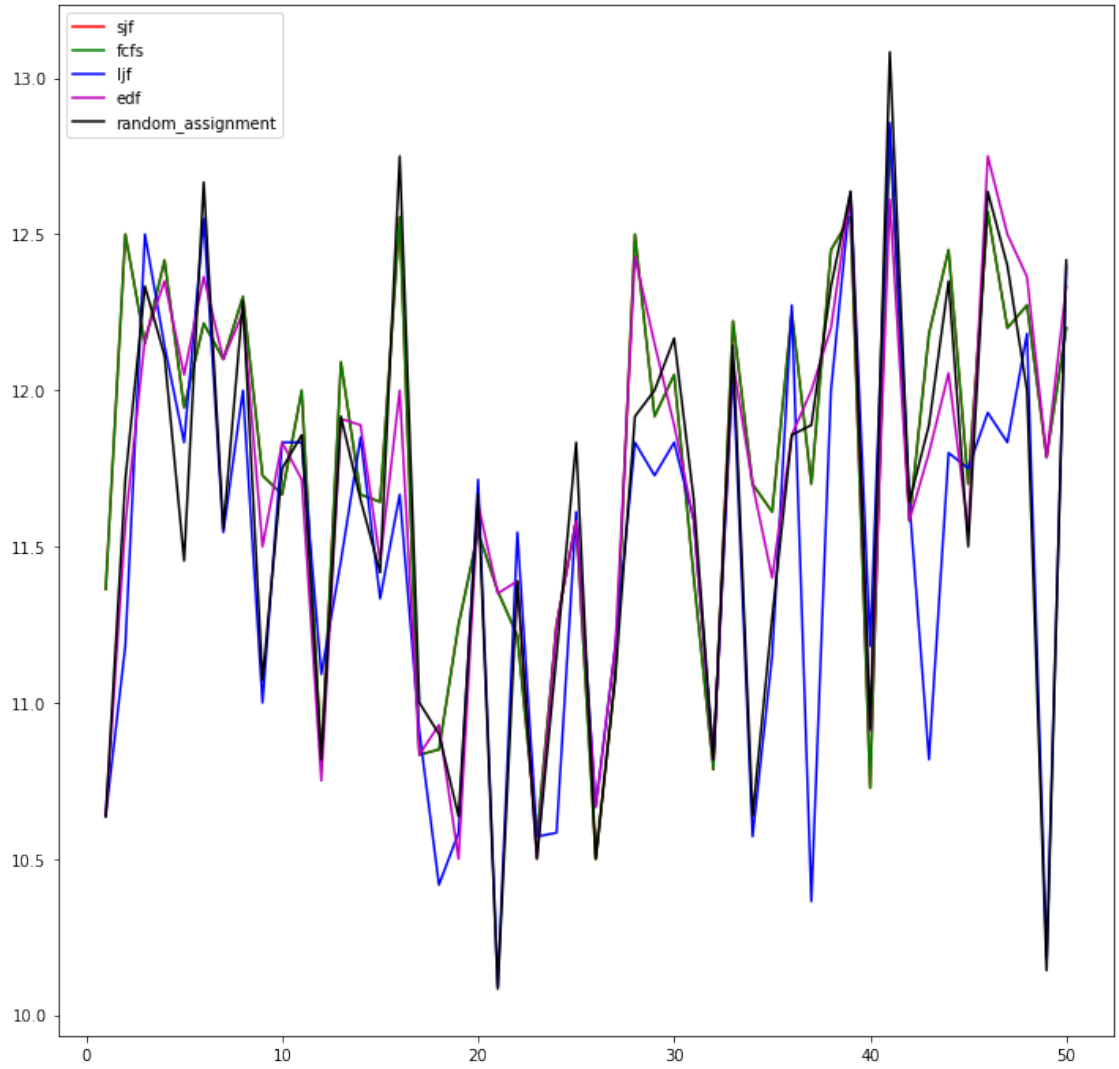
[13]:
```
def simulate_and_plot(no_of_simulations):
    sjf_result = []
    fcfs_result = []
    ljf_result = []
    edf_result = []
    random_assignment_result = []
    x_idx = np.arange(1,no_of_simulations+1,1)
    workers = generateworker(worker_no)
    for i in range(no_of_simulations):
        test = generate(cluster_no,total_task,arrival_once)
        sjf_result.append(analyze(sjf(test,workers)))
        fcfs_result.append(analyze(fcfs(test,workers)))
        ljf_result.append(analyze(ljf(test,workers)))
        edf_result.append(analyze(edf(test,workers)))
        random_assignment_result.
↪append(analyze(random_assignment(test,workers)))
    plt.figure(figsize=(12,12))
    plt.plot(x_idx,sjf_result,color='r',label='sjf')
    plt.plot(x_idx,fcfs_result,color='g',label='fcfs')
    plt.plot(x_idx,ljf_result,color='b',label='ljf')
    plt.plot(x_idx,edf_result,color='m',label='edf')
    plt.plot(x_idx,random_assignment_result,color='k',label='random_assignment')
    plt.legend()
```

0.1 Inputs

0.2 Different Arrival time

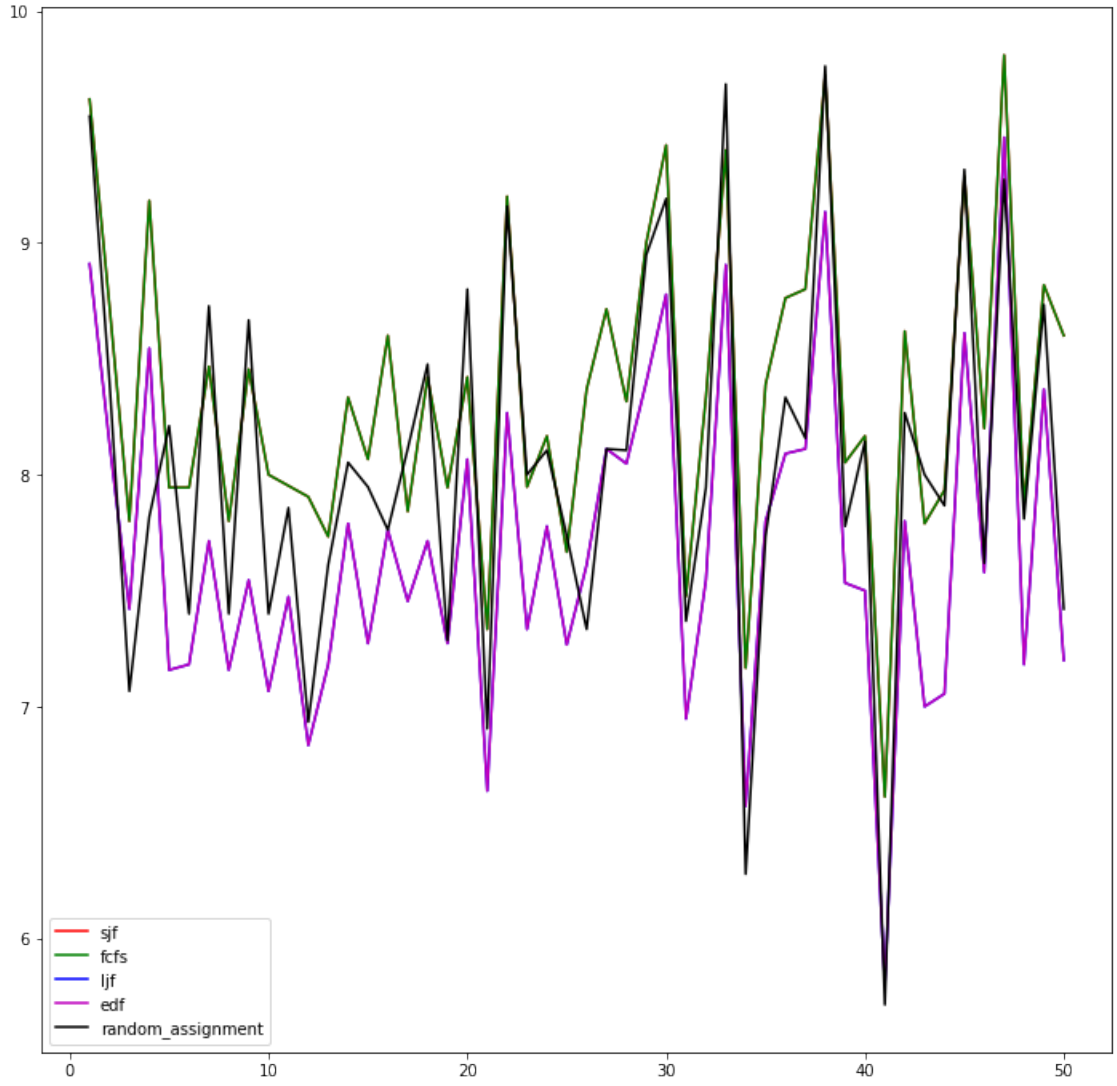
[18]:
```
cluster_no = 5
task_per_cluster = 10
total_task = cluster_no * task_per_cluster
worker_no = 5
arrival_once = False
```

```
simulate_and_plot(50)
```



0.3 All arrive at once

[19]:
```
cluster_no = 5
task_per_cluster = 10
total_task = cluster_no * task_per_cluster
worker_no = 5
arrival_once = True
simulate_and_plot(50)
```


10
9
8
7
6
0
10
20
30
40
50
sjf
fcfs
ljf
edf
random_assignment

[ ]: