COMP9334
Project Report
Z5004850
Tengyu Ma

In this project, a simulation of a multi-server system was implemented in Python. The code was generate under the Python 3.5 with pylab installed.

1. Introduction of Design

According to the instruction of the multi-server system, preprocessor and subserves obey the first-come-first-serve queueing discipline and all of them has infinite buffer, which means during the simulation period, no task would be decline. So a simple way to implement the simulation would be just handle each arrival request and calculate the finish time. If any request go over the hole simulation time, the request will not take into consideration as it is not finished. In other word, if we handle the previous finish time of preprocessor and subserves, each request can be calculate independently.

Formula Design

Internal arrive time is defined as the sum of two part. The sequence a1k is exponentially distributed with a mean arrival rate 0.85 requests per time unit. The sequence a2k is uniformly distributed in the interval [0.05, 0.25]. I use the formula in sim_mm1 :

-math.log(1-random.random())/rate

to generate exponentially distributed sequence where random.random() cloud generate random number in range[0,1):

random.uniform(0.05,0.25)

Previous function could generate uniformly distributed sequence, where uniform could generate uniform distribute in specific range. Inter_arrival time is sum of those two random number

Subserves process time is the inverse function of the The probability density function f(ts). First the integral of the function should be calculated and then we could write the formula as following:

(10.3846/(n1.65))/((1-random.uniform(0,1))**(1/2.08))**

Here random.uniform(0,1) is used to simulate Pareto distribution in python, as there is no build function to generate it. But it could be implement by uniform distribution between 0 and 1. Thus, I use random.uniform(0,1).

Work flow Design

When a request come into the system, a parameter is used to record the arrival time. While the arrival time is smaller than whole simulation period, we keep handle the request. Then we made a judgement whether the preprocessor is busy. Preprocessor was designed as an integer to record previous task's finish time. If the preprocessor is greater than current request's arrival time, preprocessor is busy. So current task will finish its preprocess in

preprocessor+preprocess_time

Otherwise current task will finish in

Arrival_time + preprocess time

After preprocess, the whole task is distribute into n sub_tasks and send to each subserves. Here a python build in function **random.sample((0,10),n)** is used to generate a randomly list of subserves distribution without repeat(we assume preprocessor do not repeatedly forward to subserves for one request). Subserves are designed as Array in order to record their previous task's finish time:

sub_servers = [0]*10

In each subserves we did same thing as what we had done for preprocessor. Comparing the value of subserves and renew the subserves. The finish time of a request is the longest finish time in each subserves. In this way, each request could be calculate independently.

The result of the simulation is stored in a list. The graph of the simulation or mean response time could be generate from the list.

In order to make the simulation reproducible, I use fake random which could generate same random number in different test.

2. Output Analysis

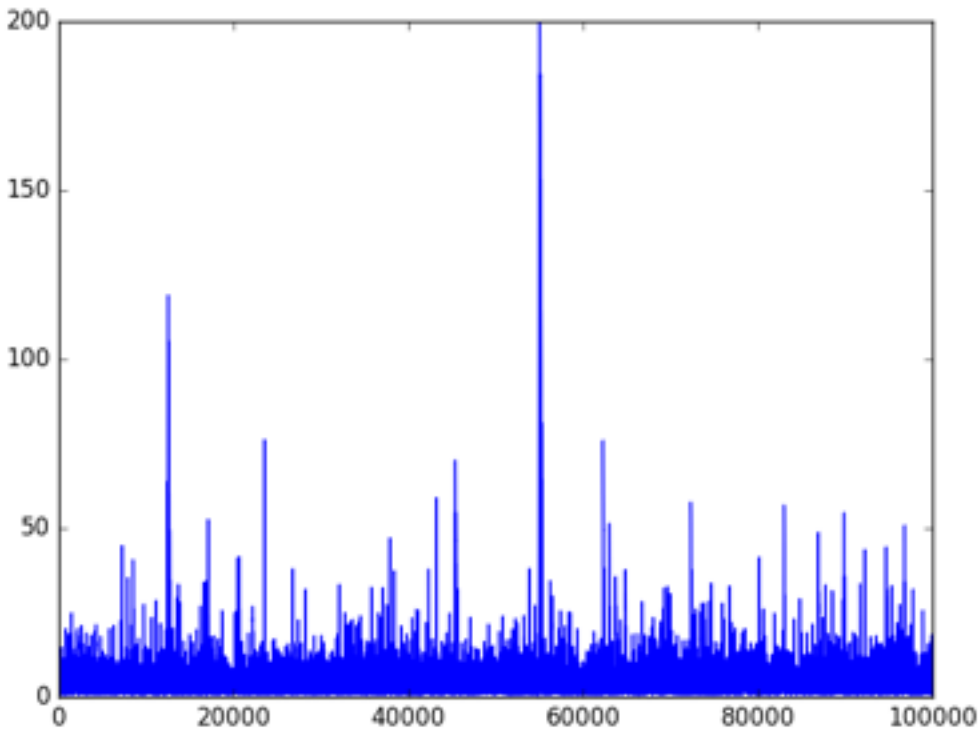
When set the whole simulation time as 10000, 80000, 90000, 100000and 200000 time unit, we could get following data set:

n	10000	80000	90000	100000	200000	300000
1	1584.1545225 50526	13646.969993 770357	15397.080720 752654	17192.30246 064073	34323.36268 422478	51349.6708340

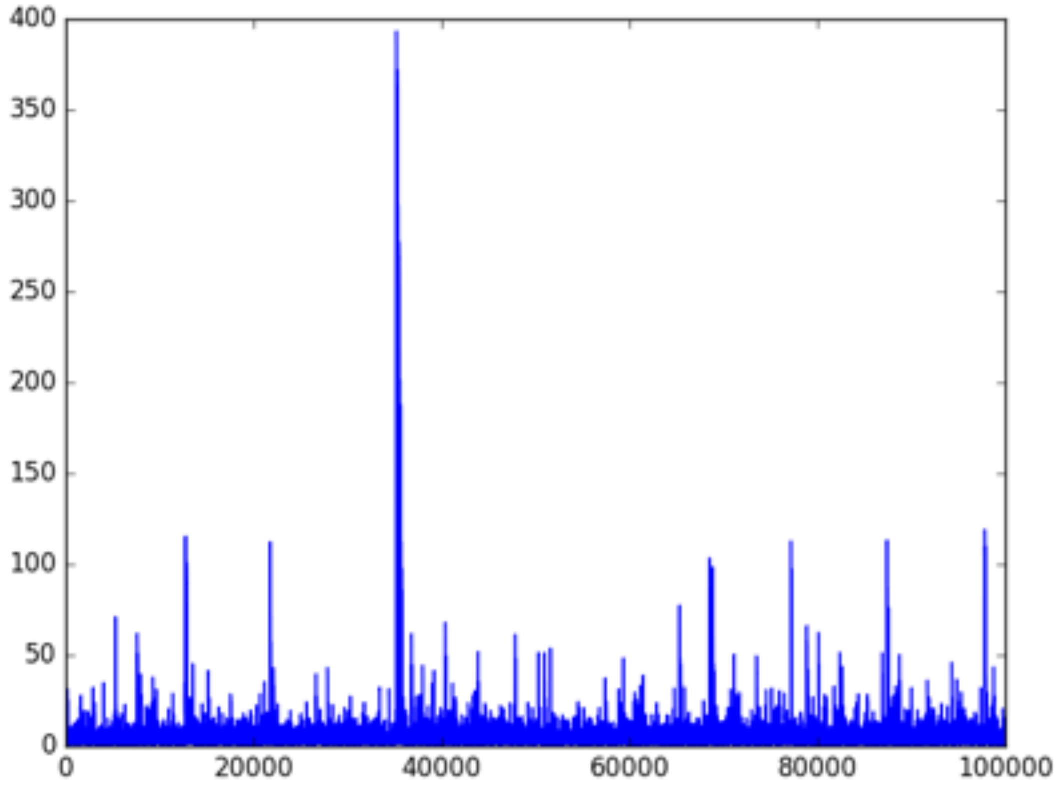
2	142.91670579648658	300.32539670816595	273.6203499062273	258.0213212923813	280.3624003812613	328.2582478097392
3	28.98719547080968	31.58763927024156	31.014511132401942	31.65077172174367	4.340152383533	33.31181835073347
4	14.36963384552062	15.931379480487541	16.040043226215605	17.07701258357501	16.18986494608148	17.198785895210353
5	10.571561664603573	18.517094002257593	9.984622738416883	9.743526070379309	12.709117265624542	28.490819813913117
6	7.602138210854852	8.137503520667755	.5591854178997	8.358498062145816	10.845516579330193	8.772087243548873
7	7.177987490645033	8.873582947636786	7.193646024397184	7.445295893778244	8.975120302817741	8.981218331644696
8	6.042401665803292	7.002578436898714	6.949917505644183	7.252334583016475	6.668591299962	6.879488979911506
9	.6933075440542	6.769851398425083	6.195669324703803	7.042908680561254	8.000783066363867	6.860541980789675
10	6.324272292759348	7.328962274990749	8.706346882506097	8.512071879948845	7.594867188841812	7.348871239605482

It is clear that although there are some fluctuate, when test time is long enough, the simulation will generate the solution that when break into 8 or 9 sub-tasks ,the mean response time would be least. Generally, when distribute tasks into to many subtasks, there is a bottle neck on preprocessor as it will spend more time when breaking task. So there is a balance point for breaking tasks, too many subtask or too small separation will lead to longer response time.

In order to make a solution of which distribution is better, we should do further test.



Take n=9 and simulation time = 100000 time unit



Take n=8 and simulation time = 100000 time unit

We could find there are very few results which are extremely huge. According to test result diagram we can set the confident interval as results which is less than 50 as most of the data distribute in this area. So we could generate new data set for n=8 and n=9 to analysis which choice is better:

n	10000	80000	900000	100000	200000	300000
8	5.824452698099063	5.742732056222329	5.737556093064794	5.791103215949062	5.748870647473025	5.867649837149568
9	5.838954234828516	6.079313128693893	5.946730086436914	6.039459659539555	5.908259323114307	6.0135865172902525

New test result shows that when we ignore result greater than 50, each test case shows that the n=8 is a better choice. In this case we could say that distribute request into 8 sub-tasks will be a better choice