COMP 9334

Capacity Planning of Computer Systems and Networks

Project 2018s1

Server setup in data centres

Report

Xiaowei Zhou

Z5108173

UNSW Sydney

Table of Contents

Abstract

This report aims at determining proper delayedoff interval for M/M/k/setup/delayedoff system to reduce mean response time of the system. To come out proper delayedoff time interval, multiple times of simulation are run for different setup of delayedoff time. Results are analysed by student-t distribution and hypothesis testing.

Server Structure

The server system is setup as 5 servers, setup time 5, arrival rate 0.35, service rate 1, and variable delayedoff time.

The aim is to configure the system to have mean response time 2 units less than baseline: delayedoff time 0.1.

Trace Simulation

The program contains a trace mode simulation function which takes in a list of arrival time and service time of jobs. Code in sim.py function traceSimulation.

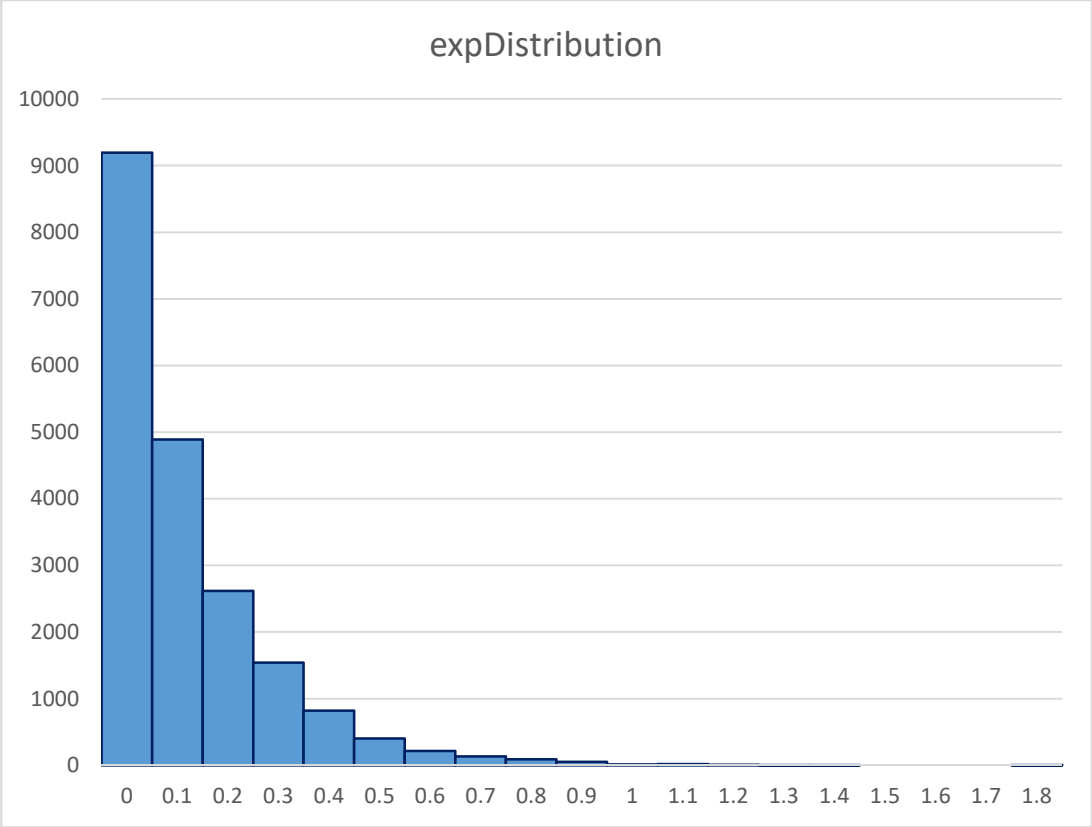
Random Simulation

The program contains two versions of random simulation: one is run directly, another is building arrival and service time list and use trace simulation. Both versions produce same result and to improve efficiency of simulation, build trace version is finally used.

These two versions are in sim.py function randomSimulation and buildTrace.

In random simulation, exponential distribution is used for arrival time and service time.

To generate proper numbers, -log(1-random())/rate is used, while random() generates uniform distributed numbers in 0 to 1. Arrival times is simply exponential distributed while service time is a sum of three independent exponential distributed numbers. To ensure the generated random numbers are correct, code expDistributionTest.py (under root directory) is used and by running it, 20000 random numbers for exponential distribution are generated with rate 6, writing to expDis.csv. The graph below is constructed from the csv file, showing a proper exponential distribution.



Simulation Correctness

As both random and trace simulations use traceSimulation function, and buildTrace function has been proved in section Random Simulation, the correctness is based on traceSimulation function itself.

Trace simulation itself contains a proper logic that next event will be one of the following four: an arrival, a departure, a server finish setup, a server shut down. A simple trace configuration is used for testing: 2 server, setup time 5, delayed off time 5,

jobs arrive at [5,6,7,8,9,10,11,12,13,14,15] and service time are all 0.5. The result of simulation shows the following departures (any words in brackets is addon description and not produced by simulation) :

5.000 10.500 (Job 1, start server 1 at 5, until 10 to process, 10.5 finish)
6.000 11.000 (Job 2, start server 2 at 6, but taken by server 1, 11 finish)
7.000 11.500 (Job 3, queue and take by server 1, 11.5 finish)
8.000 11.500 (Job 4, queue and take by server 2 at the same time, 11.5 finish)
9.000 12.000 (Job 5, queue and take by server 1, 12 finish)
10.000 12.000 (job 6, queue and take by server 2 at the same time, 12 finish)
11.000 12.500 (job 7, queue and take by server 1, 12.5 finish)
12.000 12.500 (job 8, directly come in and take by server 2, 12.5 finish)
13.000 13.500 (job 9, directly come in and take by server 1, server 2 is in delayedoff)
14.000 14.500 (job 10, directly come in and take by server 1)
15.000 15.500 (job 11, directly come in and take by server 1)
Note. This test can be found under directory sample/, This is test 3.

It is clear that servers are configured all OFF at time 0, start their first setup at when the first job arrival, and response time finally goes to steady state 0.5.
Besides, setup time, delayedoff time, departure time for each server are set to sys.maxsize in python by default.

The simple test from sample input/output files in Section 5.4 in project spec also proved that the whole trace simulation is correct – all same results after simulation. (These two tests can be found under directory sample/, these are test 1 and 2.)

Reproducibility

Simple trace simulation is always reproducible. For random simulation, reproducibility for this simulation program depends on same configuration including the test index. For example, same configuration for test 1 and test 2 will produce different results since the random seed selected is based on test index.

This is designed to help with massive times of simulation with same configuration to find out proper delayedoff time. To check reproducibility, test index is important.
A sample input/output file set is stored in directory reproducibility/.

Figure out delayedoff time

To figure out proper delayedoff time, simulation for different delayedoff time to get their mean response time is needed. To make results accurate and confident, the following procedure and configuration are used:

5 servers
Setup time 5
End time 5000
Arrival rate 0.35
Service rate 1
Mode random
Delayedoff time: [0.1, 2.5, 5.0, 7.5, 10.0, 11.0, 12.0, 13.0, 14.0]
Replications: 30 times for each delayedoff time
Transient: first 25% of all jobs processed

Besides output of mean response time and departure list, another script Tdistribution.py will apply student-t distribution for 30 MRT results and come up a mean, a std, and 95% confidence interval.

For each simulation, end time 5000 is chosen to ensure simulation is long enough for system go into steady state. By an end time of 5000, approximately 1735 jobs departure for each single simulation on average, and the first 25% of them are considered as transient. These transient part data are not counted in MRT returned by simulation function but do record in departure_*.txt.

30 replications are the lower limit of Central Limit Theorem that allowed a hypothesis testing to check if the current delayedoff time setup is confidently 2 units less than baseline setting 0.1 delayedoff time.

The statistical results for each delayedoff time are listed below:

Delayedoff Time	MRT	Std Dev.	95% CI
0.1	6.063	0.074	(5.911, 6.214)
2.5	5.227	0.088	(5.046, 5.407)
5.0	4.666	0.066	(4.531, 4.802)
7.5	4.281	0.062	(4.155, 4.407)
10.0	4.025	0.062	(3.899, 4.151)
11.0	3.948	0.066	(3.813, 4.082)
12.0	3.877	0.066	(3.742, 4.011)
13.0	3.819	0.066	(3.684, 3.954)
14.0	3.765	0.059	(3.643, 3.886)

These data are recorded in directory results/ and named DelayedoffTimeDO(e.g. 0.1DO, 10.0DO)
Note. Configuration and output files are not stored to avoid submission packet limit, and these results can be reproduced easily.

To test if a delayedoff time is confidently 2 units less than baseline setting, use hypothesis testing:

H0: current delayedoff time is not 2 units less than baseline setting
HA: current delayedoff time is 2 units less than baseline setting
Significant = 0.5, 95% CI used
For baseline setting with 2 units less, it becomes mean 4.063, std 0.074, 95% CI (3.911, 4.214)

Thus, the first delayedoff time that reject H0 is 12.0 (with 95% CI overlap) and 14.0 (with 95% CI not overlap).

Conclusion

Therefore, delayedoff should be set to 12.0 (accept CI overlap) or 14.0 (no CI overlap) to make MRT 2 units less than baseline setting.