COMP9334 Project, Session 1, 2018: Server setup in data centres

Version 1.1
Due Date: 11:00pm Sunday 20 May 2018.

This version: 22 April 2018

Updates to the project, including any corrections and clarifications, will be posted on the subject website. Make sure that you check the course website regularly for updates.

Change log

Note: New text is shown in red coloured font. Deleted text is retained and shown as strikethrough, e.g. ~~this is deleted text~~.

- Version 1.1. (22 April 2018) Changes in: Table 4, Section 3.1.2, Section 5.2.3.

1 Introduction and learning objectives

This project is inspired by the research work reported in the article *Exact analysis of the M/M/k/setup class of Markov chains via recursive renewal reward* [1]. The paper studies a dilemma faced by data centre operators on trading off energy consumption and latency. The key question that the paper asks is whether one should turn an idling server off. The advantage of turning off an idling server is that it can save a lot of energy. However, if an off server is needed again, it has to be setup and this adds latency to job processing. In this project, you will use simulation to try to answer a similar research question.

In this project, you will learn:

1. To apply discrete event simulation to a performance analysis problem
2. To use statistically sound methods to analyse simulation outputs

2 Support provided

If you have problems doing this assignment, you can post your question on the message board on the subject website. **We strongly encourage you to do this as asking questions and trying to answer them is a great way to learn. Do not be afraid that your question may appear to be silly, the other students may very well have the same question!**

3 Description of the computer system

Figure 1 depicts the computer system to be considered in this project. The system consists of a dispatcher as its front-end and m servers at its back-end. There is only one queue and it is located at the dispatcher; there are no queues at the computer servers. You can assume that:

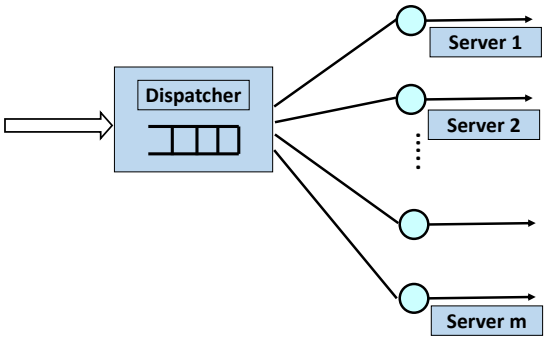

Figure 1: Architecture of a computer system. The front-end consists of a dispatcher and the back-end has a number of computer servers. There is a queue at the dispatcher. There are no queues at the servers.

- The dispatcher has sufficient memory so the number of waiting room for queueing can be considered to be infinite.
- The communication between the dispatcher and the server is fast. This means that once a server has finished working on a job, it can inform the dispatcher immediately. It also means that it takes the dispatcher negligible time to send a job to a server.
- The dispatcher takes negligible time to make a decision.

We discussed the multi-server queue in Week 3's lecture. In the setup in Week 3, all servers remain on at all times. A server can only have two states: busy or idle. An arriving job will be served immediately if there is an idling server, otherwise it will join the queue at the dispatcher. Once a server has finished processing a job, it will serve the job at the front of the queue if there is one. In particular, if the inter-arrival time and service time probability distributions are exponential, then we learnt in Week 3 that this type of multi-server queue could be modelled as an M/M/m queue.

Energy consumption is a major issue faced by data centres. A drawback of the multi-server queue described in the last paragraph is that the idling servers consume power but are not doing useful work. There have been various proposals to reduce the power consumption and they are discussed in Section 3 of [1]. We recommend that you go through Sections 1 and 3 of [1] to get a better understanding of the various proposals because reading those sections will help you to put the project work into context. For this project, you will focus on the *setup/delayedoff* mode of operation which is similar (but **not** identical) to that discussed in Section 3.2 of [1].

3.1 The *setup/delayedff* system

We will now explain the *setup/delayedoff* mode of operation that is used in this project.

3.1.1 Server state

In this project, a server can have four states:

- The OFF state means the server is powered off. A server in the OFF state cannot process a job.
- If a server in the OFF state is powered on, then it enters the SETUP state. In this state, the server is in the process of booting up. A server in the SETUP state cannot process a job. We will use the term **setup time** to refer to the time needed to boot up the server; this is the time from the moment the server is switched on to the moment the setup is finished. Once the setup is finished, the server can start to process jobs. We assume in this project that the setup time is deterministic.
- The BUSY state means the server is processing a job.
- After a server in the BUSY state has finished processing a job, it will check whether there is a job waiting at the dispatcher. If the dispatcher queue is empty, the server will enter the DELAYEDOFF state. When a server enters the DELAYEDOFF state, it will start a countdown timer. The initial value of the countdown timer is denoted by T_c which is a positive value. During the DELAYEDOFF state, the server is still powered on and the dispatcher *may* send a job to a server in the DELAYEDOFF state to process. There are two possibilities to consider, depending on whether the dispatcher sends a job to the sever in the DELAYEDOFF state. Let us discuss these two possibilities separately:
  - If the dispatcher does not send any jobs to a server in the DELAYEDOFF state before the countdown timer expires (i.e. reaches the value of zero), then the server will power off itself when the countdown timer expires. In other words, at the time when the countdown timer expires, the server will go from DELAYEDOFF state to OFF state.
  - If the dispatcher sends a job to a server in the DELAYEDOFF state before the countdown timer expires, the server will change its state to BUSY and process the job. The countdown timer will be removed. Note that a server may cycle between the BUSY and DELAYEDOFF states many times; each time when the server goes from BUSY to DELAYEDOFF state, the countdown timer is initialised to T_c.

3.1.2 Handling arrivals

Having described the four possible server states, we will now explain what happens when a job arrives at the dispatcher. It is important to remember that the dispatcher cannot send a job to a server in OFF, SETUP or BUSY state. However, the dispatcher can send a job to a server in the DELAYEDOFF state for processing. It can also send a job to a server that has just finished setting up (i.e. the end of SETUP) or finished processing a job; these two possibilities will be discussed later on.

When a job arrives at the system, the dispatcher will use the following rules:

1. If there is at least a server in the DELAYEDOFF state, then the dispatcher will send the arriving job to a particular server in the DELAYEDOFF state. The choice of the server depends on the value of the countdown timer of the servers at the time when the job arrives at the dispatcher. The dispatcher will send the job to the server with the **highest** value in the countdown timer. The selected server will cancel its countdown timer and change its state to BUSY.
2. If there are no servers in the DELAYEDOFF state, then the dispatcher will check whether there are servers in the OFF state. If there is at least a server in the OFF state, then the dispatcher will select one of them and turn it on. There are no specific criteria to choose an OFF server, so any one of them can be chosen. The state of the selected server will change

from OFF to SETUP. The dispatcher will put the job ~~in~~ at the end of the queue and **mark** the job. The jobs in the queue can be either MARKED or UNMARKED. A job in the queue is MARKED if it is waiting for a server to be setup. Hence, a simple consistency check is that the number of MARKED jobs in the queue should equal to the number of servers in the SETUP state.

3. If there are no servers in the DELAYEDOFF state and there are no servers in the OFF state, then it means all the servers are either in BUSY or SETUP state. In this case, the job is put ~~in~~ at the end of the queue and is UNMARKED.

3.1.3 Handling departures

We now describe what happens when a job departs from a server. There are two cases to consider depending on whether there is a job in the queue in the dispatcher. In the following description, the server is referring to the server that has just completed processing of a job.

1. If the dispatcher queue is empty, then the server will change its state from BUSY to DELAYEDOFF. It will also start the countdown timer. The initial value of the countdown timer is T_c. In this project, you can assume that T_c is deterministic.

2. If there is at least one job at the queue, the server will take the job from the head of the queue. The state of the server remains BUSY. The other actions of the dispatcher depends on whether the job that has been sent to the server for processing is MARKED or UNMARKED.

   (a) If the job is UNMARKED, then it means this job is not waiting for a server to be setup. There is no further action to be taken.

   (b) If the job is MARKED, then it means this job is waiting for a server which is still in the process of setting up. Since the dispatcher is sending this job to a server which is different from the one that the job is waiting for, the dispatcher has to decide whether the set up process should continue. The rationale is that we do not wish to complete the setup and find that the newly setup server has no job to process. In this case, the dispatcher will check whether there is an UMARKED job in its queue. If there is one, it means there is a job in the queue which is not associated with a server in the SETUP state. The dispatcher's actions are:

      i. If there is at least a UNMARKED job, then the dispatcher will choose the first UMARKED job and change it to a MARKED job.

      ii. If there are no UNMARKED jobs, then the dispatcher will need to turn off a server that is in the SETUP state. In this case, the dispatcher will look at how much time the servers in the SETUP state still need to complete setting up and it will choose to turn off the server with the longest remaining setup time. The state of the selected server will change from SETUP to OFF and you can assume that this change of state is immediate. You may ask how the dispatcher knows what the remaining setup time is. Note that setup starts when a job arrives at the dispatcher and there is an OFF server, so the dispatcher knows the time setup begins. It also knows the time when setup finishes because we assume setup time is deterministic.

3.1.4 Other events

Now we have finished describing all arrival and departure events. There are two more possible events in this system.

1. One event is a server has finished its setup. This server will take a MARKED job off the queue. The status of the server will change from SETUP to BUSY.

2. Another event is the expiry of the countdown timer of a server in DELAYEDOFF state. The status of the server will change from DELAYEDOFF to OFF.

Note that for M/M/1 type of queues that we discussed in the lecture, there are only two types of events: arrival and departure. However, for the *setup/delayedoff* queue, there are also the two types of events discussed above.

3.1.5 Initial condition

You should assume that the initial condition of the system is: all servers are OFF and the dispatcher queue is empty.

3.2 Examples

We will now present two examples to illustrate the operation of the *setup/delayedoff* system.

3.2.1 Example 1

The aim of this example is to illustrate the setting up of servers, shutting down a server that is being setup and changing a job in the dispatcher from UNMARKED to MARKED.

In this example, there are $m = 3$ servers. The arrival and service times of the jobs are shown in Table 1. We assume all servers are in the OFF state at time zero. The setup time is assumed to be 50. The initial value of the countdown timer is $T_c = 100$. Table 2 shows the on-paper simulation with explanatory comments.

Arrival time	Service time
10	1
20	2
30	3
33	4

Table 1: Example 1: Job arrival and service times.

Master clock	Dispatcher	Server 1	Server 2	Server 3	Notes
$t = 0$	–	OFF	OFF	OFF	Initially, dispatcher is empty. All servers are OFF
$t = 10$	(10,1,MARKED)	SETUP (complete at t = 60)	OFF	OFF	An arrival at time 10. Server 1 is turned on and is in SETUP state. The jobs in the dispatcher are modelled with 3-tuples: first element is the arrival time, second element is the service time and the last element indicates whether it is MARKED or UNMARKED. This job is MARKED because it is waiting for the setting up of a server.
$t = 20$	(10,1,MARKED) (20,2,MARKED)	SETUP (complete at $t = 60$)	SETUP (complete at $t = 70$)	OFF	An arrival at time 20. Server 2 goes into SETUP state.
$t = 32$	(10,1,MARKED) (20,2,MARKED) (32,3,MARKED)	SETUP (complete at $t = 60$)	SETUP (complete at $t = 70$)	SETUP (complete at $t = 82$)	An arrival at time 32. Server 3 goes into SETUP state.
$t = 33$	(10,1,MARKED) (20,2,MARKED) (32,3,MARKED) (33,4,UNMARKED)	SETUP (complete at $t = 60$)	SETUP (complete at $t = 70$)	SETUP (complete at $t = 82$)	An arrival at time 33. All servers are in SETUP state. The job is UNMARKED because it is not associated with a server in SETUP state.
$t = 60$	(20,2,MARKED) (32,3,MARKED) (33,4,UNMARKED)	BUSY (10,1)	SETUP (complete at $t = 70$)	SETUP (complete at $t = 82$)	Setup of server 1 completed. Server 1 processing the job that arrives at time 10 and needs a processing time of 1. Server 1 state is now BUSY.
$t = 61$	(32,3,MARKED) (33,4,MARKED)	BUSY (20,2)	SETUP (complete at $t = 70$)	SETUP (complete at $t = 82$)	Server 1 completes the job (10,1). It takes the job from the front of the queue which is (20,2,MARKED). This job is MARKED which means it is waiting for a server to SETUP. The last job in the queue was UNMARKED so this job is now MARKED.
$t = 63$	(33,4,MARKED)	BUSY (32,3)	SETUP (complete at $t = 70$)	OFF	Server 1 completes the job (20,2). It takes the job from the front of the queue which is (32,3,MARKED). This job is MARKED which means it is waiting for a server to SETUP. There are no more UNMARKED jobs in the queue so the dispatcher chooses to turn off Server 3 which has the longest residual setup time.
$t = 66$	–	BUSY (33,4)	OFF	OFF	Server 1 completes the job (32,3). It takes the job from the front of the queue which is (33,4,MARKED). This job is MARKED which means it is waiting for a server to SETUP. Server 2 is now OFF.
$t = 70$	–	DELAYEDOFF expires $t = 170$	OFF	OFF	Server 1 completes the job (33,4). Queue is empty. Server 1 changes state to DELAYEDOFF and the countdown timer will expire at $t = 170$.
$t = 170$	–	OFF	OFF	OFF	Countdown timer expires and Serve 1 goes to OFF state.

Table 2: Table illustrating the updates for Example 1.

3.2.2 Example 2

The aim of this example is to illustrate the situation when there are multiple servers in the DELAYEDOFF state.

In this example, there are $m = 3$ servers. In order to shorten the description, we will start from time 10 and the state of the system at this time is shown in Table 4. The arrival and service times of the job after time 10 are shown in Table 3. The setup time is assumed to be 5. The initial value of the countdown timer is $T_c = 10$. Table 4 shows the on-paper simulation with explanatory comments.

Arrival time	Service time
11	1
11.2	1.4
11.3	5
13	1

Table 3: Example 2: Job arrival and service times.

Master clock	Dispatcher	Server 1	Server 2	Server 3	Notes
$t = 10$	–	DELAY-EDOFF expires $t = 20$	DELAY-EDOFF expires $t = 17$	OFF	We begin with time 10. The dispatcher queue is empty.
$t = 11$	–	BUSY (11,1)	DELAY-EDOFF expires $t = 17$	OFF	An arrival at time 11. Server 1's countdown timer expires at 20 and Server 2's expires at 17. Server 1 is chosen because it has a longer residual time to shutdown. Server 1 changes state from DELAYEDOFF to BUSY.
$t = 11.2$	–	BUSY (11,1)	BUSY (11.2,1.4)	OFF	An arrival at time 11.2. Server 2 changes state to BUSY.
$t = 11.3$	(11.3,5,MARKED)	BUSY (11,1)	BUSY (11.2,1.4)	SETUP (to be completed at t =16.3)	An arrival at time 11.3. Server 3 changes state to SETUP.
$t = 12$	–	BUSY (11.3,5)	BUSY (11.2,1.4)	OFF	Server 1 finishes processing the last job. Queue not empty. Server 1 takes first job in the queue. This job is a MARKED job. No jobs are queueing behind this job. Server 3 stops setup and is turned OFF.
$t = 12.6$	–	BUSY (11.3,5)	DELAY-EDOFF expires $t = 22.6$	OFF	Server 2 finishes processing. Queue empty. Server 2 changes state to DELAYEDOFF.
$t = 13$	–	BUSY (11.3,5)	BUSY (13,1)	OFF	An arrival at time 13. Server 2 changes state to BUSY.
$t = 14$	–	BUSY (11.3,5)	DELAY-EDOFF expires $t = 24$	OFF	Server 2 changes state to DELAYEDOFF. Note that the countdown timer starts from T_c again.
$t = 17$ ~~16.3~~	–	DELAY-EDOFF expires $t = 27$ ~~26.3~~	DELAY-EDOFF expires $t = 24$	OFF	Server 1 finishes processing. Queue empty. Server 1 changes state to DELAYEDOFF.
$t = 24$	–	DELAY-EDOFF expires $t = 27$ ~~26.3~~	OFF	OFF	Countdown timer for Server 2 expires. Server 2 changes state to OFF.
$t = 27$ ~~26.3~~	–	OFF	OFF	OFF	Countdown timer for Server 1 expires. Server 1 changes state to OFF.

Table 4: Table illustrating the updates in Example 2.

4 Project description

This project consists of two main parts. The first part is to develop a simulation program for the *setup/delayedoff* system which is described in Section 3. In the second part, you will use the simulation program that you have developed to solve a design problem.

4.1 Simulation program

You must write your simulation program in one (or a combination) of the following languages or numerical package: Python (either version 2 or 3), C, C++, Java or Matlab. All these languages and numerical packages are available on the CSE system. We will test your program on the CSE system so your submitted program **must** be able to run on a CSE computer. Note that it is possible that due to version differences, code that runs on your own computer may not work on the CSE system. It is your responsibility to ensure that your code works on the CSE system.

We require you to write your simulation program as a **function** in your chosen language. Your function must have the following seven inputs:

1. A parameter `mode` of string type. This parameter is to control whether your program will run simulation using randomly generated arrival and service times, or in trace driven mode. The value that the parameter `mode` can take is either `random` or `trace`.
2. A parameter `arrival` for supplying arrival information to the program. The meaning of `arrival` depends on `mode`. We will provide more information later on.
3. A parameter `service` for supplying service time information to the program. The meaning of `service` depends on `mode`. We will provide more information later on.
4. A parameter `m` which is the number of servers. This parameter is a positive integer.
5. A parameter `setup_time` for the setup time. This parameter is a positive floating point number.
6. A parameter `delayedoff_time` for the initial value of the countdown timer T_c. This parameter is a positive floating point number.
7. A parameter `time_end` which stops the simulation if the master clock exceeds this value. This parameter is only relevant when `mode` is `random`. This parameter is a positive floating point number.

Note that your simulation program must be a general program which allows different values for the number of servers, setup time, T_c and simulation end time to be used. When we test your program, we will vary the parameter values. You can assume that we will only use valid inputs for testing.

You can have additional parameters if you wish.

4.1.1 The random mode

When your simulation is working in the `random` mode, it will generate the **inter-arrival** times and service times in the following manner.

1. The inter-arrival probability distribution is exponentially distributed with parameter λ. This means the mean arrival rate of the jobs is λ. You will need to supply the value of λ to your program using the input parameter `arrival`.

2. Let s_k denote the service time of the k-th job arriving at the dispatcher. Each s_k is the sum of three random numbers s_{1k}, s_{2k} and s_{3k}, i.e $s_k = s_{1k} + s_{2k} + s_{3k}$ $\forall k = 1, 2, ...$ where s_{1k}, s_{2k} and s_{3k} are exponentially distributed random numbers with parameter μ. You will need to supply the value of μ to your program using the input parameter `service`.

   The easiest way to generate the service time is to generate three independent exponentially distributed numbers and add them up. If you add independent exponentially distributed random variables, you get phase type distribution, see the text by Harchol-Balter.

4.1.2 The trace mode

When your simulation is working in the `trace` mode, it will read the list of arrival times and list of service times from two ASCII files.

Let us use Example 1 in Section 3.2.1 for an illustration. The arrival times are $[10, 20, 32, 33]$ and the service times are $[1, 2, 3, 4]$. Your program is required to simulate until all jobs have departed. For this example, the last departure occurs at time 70 and you can stop the simulation there.

We will supply the arrival times and service times to you using two ASCII files. The names of these two files have specific format, which we will discuss in Section 5 on testing. For Example 1, the arrival times will be specified by a file whose contents are as follows:

```
10
20
32
33
```

where each arrival time takes up one line of the file. The same format is used for service times. You can assume that within a test, the number of arrival times and the number of service times are equal.

4.2 Determining a suitable value of T_c

After writing your simulation program, your next step is to use your simulation program to do a design.

For this design problem, you will assume the following parameter values: the number of servers is 5, setup time is 5, $\lambda = 0.35$, $\mu = 1$.

Let us assume that your *baseline* system uses $T_c = 0.1$. This baseline system will give you poor response time because the servers have to be powered up again frequently.

Your aim is to design an *improved* system which uses a higher value of T_c so as to reduce the response time. Your aim is to determine a value of T_c (or a range for T_c) so that the improved system's response time must be 2 units less than that of the baseline system.

In solving this design problem, you need to ensure that you use statistically sound methods to compare systems. You will need to consider parameters such as length of simulation, number of replications, transient removals and so on. You will need to justify in your report how you choose the value of T_c for the improved system.

5 Testing simulation program

As part of the assessment for this project, you are asked to **attend an interview** so that we can test your simulation program. During the interview, we will ask you to run a series of tests. Each

test is specified by a number of configuration files. For each test, we require two output files of specific format so that we can verify the correctness of your simulation program. The aim of this section is to specify the format of the configuration files and output files.

The number of tests that you will need to run is specified in an ASCII file with name `num_tests.txt`. If we want to run 12 tests in the interview, the file `num_tests.txt` contains the string `12` only.

Each test is specified by 4 configurations files. We will index the tests from 1. If 12 tests are used, then the indices for the tests are 1, 2, ..., 12. The names of the configuration files are:

- For Test 1, the configuration files are `mode_1.txt`, `para_1.txt`, `arrival_1.txt` and `service_1.txt`. The files are similarly named for indices 2, 3, ..,9.
- For Test 10, the configuration files are `mode_10.txt`, `para_10.txt`, `arrival_10.txt` and `service_10.txt`. The files are similarly named if the test index is a 2-digit number.

We will refer to these files using the generic names `mode_*.txt`, `para_*.txt` etc.

5.1 Wrapper file

When you submit your project, you must include a **wrapper** file which will loop through all the tests. We will use this wrapper file in your interview to run your simulation. The pseudo-code for your wrapper file is as follows.

```
Read the file num_tests.txt to determine the number of tests

FOR test_index from 1 to num_tests
    Read in the configuration files for the test_index
    Set the simulation mode and parameter values
    Call your simulation function
    Write the output files % This can be in the wrapper or your simulation function
```

The filename of this wrapper file must be called `wrapper` followed by the appropriate file extension for the language that you used, e.g. `wrapper.py` for Python.

Note that another purpose of the wrapper file is to show us how to run your code. We may need to run additional tests and the wrapper file will allow us to do that. You should have comments in the wrapper file to explain to us how to run your code.

5.2 Configuration file format

5.2.1 `mode_*.txt`

This file is to indicate whether the simulation should run in the `random` or `trace` mode. The file contains one string, which can either be `random` or `trace`.

5.2.2 `para_*.txt`

If the simulation mode is `trace`, then this file has three lines. The first line is the number of servers m. The second line is the setup time. The third line is the value of T_c. If the test is Example 1 in Section 3.2.1, then the contents of this file are:

```
3
50
100
```

If the simulation mode is `random`, then this file has four lines, which contain the values of m, the setup time, T_c and `time_end`.

5.2.3 `arrival_*.txt`

The contents of the file `arrival_*.txt` depend on the mode of the test. If mode is `trace`, then the file `arrival_*.txt` contains the arrival times of the jobs with one arrival time occupying one line. For Example 1, this file will be

```
10
20
32
33
```

You can assume that the list of arrival times are in ascending order.

If the mode is `random`, then the file `arrival_*.txt` contains a string for a floating point number. This number corresponds to the value of λ.

5.2.4 `service_*.txt`

For `trace` mode, the file `service_*.txt` contains one service time per line.

For `random` mode, the file `service_*.txt` contains a string for a floating point number. This number corresponds to the value of μ.

5.3 Output file format

In order to test your simulation program, we need two output files **per test**:

1. One file containing the mean response time
2. Another file containing the departure times. Note that:
   - If the simulation is in the `random` mode, then we want the departure time of all jobs that departed at or before `time_end`.
   - If the simulation is in the `trace` mode, then we want the departure time of all jobs in the trace.

If the number of tests `num_tests` is 12, then we expect 24 files with names `mrt_1.txt`, `mrt_2.txt`, ... `mrt_12.txt` (for mean response time) and `departure_1.txt`, `departure_2.txt`, ... `departure_12.txt` (for departure times.)

The mean response time (a scalar value) should be written to a file whose filename has the form `mrt_*.txt`. For Example 1, the contents of this file are:

```
41.250
```

The departure times should be written to a file whose filename has the form `departure_*.txt`. For Example 1, the contents of the file are:

```
10.000  61.000
20.000  63.000
32.000  66.000
33.000  70.000
```

The requirements are:

1. Each line contains the arrival time and departure time of a job. The arrival time is printed first, followed by a tab, then the departure time.
2. The jobs must be ordered according to the ascending time of departure.

All numbers in `mrt_*.txt` and `departure_*.txt` should be printed as floating point numbers to exactly 3 decimal places.

5.4 Sample files

Sample files are available on the project website.

5.5 Test for reproducibility

We require that your simulation experiments are **reproducible** when operating in `random` mode. We require you to submit at least three sets of sample output files. Each set of output files should include at least two files: a file similar to `mrt_*.txt` (which contains the mean response time) and a file similar to the file `departure_*.txt` (which contains the departure times.) We will pick from these sample output files and ask you to reproduce the results in the interview.

6 Project requirements

This is an individual project. You are expected to complete this project on your own.

6.1 Submission requirements

Your submission should include the following:

1. A written report
   (a) Only soft copy is required.
   (b) It must be in Acrobat pdf format.
   (c) It must be called "report.pdf".
2. Program source code:
   (a) For doing simulation
   (b) The wrapper file, see Section 5.1
3. Sample outputs for reproducibility testing, see Section 5.5
4. Any supporting materials, e.g. logs created by your simulation, scripts that you have written to process the data etc.

The assessment will be based on your submission and the tests conducted at your interview. Note that the interview is based on the code that you submitted and will be run on a CSE computer. It is important that you submit the right version of the code and make sure that it runs on the CSE system. In the interview, we do **not** accept code that is not the submitted version and we do **not** accept code running on your computer.

It is important that you write a clear and to-the-point report. You need to aware that you are writing the report to the marker (the intended audience of the report) not for yourself. Your report will be assessed primarily based on the quality of the work that you have done. You do

not have to include any background materials in your report. You only have to talk about how you do the work and we have provided a set of assessment criteria in Section 6.3 to help you to write your report. In order for you to demonstrate these criteria, your report should refer to your programs, scripts, additional materials so that we are aware of them.

6.2 Interview

You are also required to demonstrate your work in an interview in Week 13 or StuVac. Further details on the interview will be available later. In the interview, we will ask you to run a number of tests and to demonstrate that your work is reproducible.

6.3 Assessment criteria

We will assess the quality of your project based on the following criteria:

1. The correctness of your simulation code. For this, we will:
   (a) Test your code using test cases in an interview and through additional testing if necessary
   (b) Look for evidence in your report that you have verified the correctness of the inter-arrival probability distribution and service time distribution correctly. You can include appropriate supporting materials to demonstrate this in your submission.
   (c) Look for evidence in your report that you have verified the correctness of your simulation code. You may derive test cases such as those in Section 3.2 to test your code. You can include appropriate supporting materials to demonstrate this in your submission.
2. You will need to demonstrate that your results are reproducible. We will ask you to do that at your interview. In addition, you should provide evidence of this in your report.
3. For the part on determining a suitable value of T_c, we will look for the following in your report:
   (a) Evidence of using statistically sound methods to analyse simulation results
   (b) Explanation on how you choose your simulation and data processing parameters, e.g lengths of your simulation, number of replications, end of transient etc.

6.4 How to submit

You should "tar", "rar" or "zip" your report, programs and supporting materials into a file called "project.tar", "project.rar" or "project.zip". The submission system will only accept one of these filenames.

You should submit your work via the course website. Note that the maximum size of your submission should be no more than 20MBytes.

You can submit multiple times before the deadline. The latest submission overrides the earlier submissions, so make sure you submit the correct file. Do not leave until the last moment to submit, as there may be technical or communication error and you will not have time to rectify.

7 Plagiarism

You should be aware of the UNSW policy on plagiarism. Please refer to the course web page for details.

References

[1] A. Gandhi, S. Doroudi, M. Harchol-Balter and A. Scheller-Wolf. "Exact analysis of the M/M/k/setup class of Markov chains via recursive renewal reward", Queueing Systems, 2014. https://link.springer.com/article/10.1007/s11134-014-9409-7.