COMP9334 Project, Term 1, 2024: Computing clusters

Due Date: 5:00pm Friday 19 April 2024

Version 1.01

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

Change log

- Version 1.01 (27 March 2024). There is a mistake in the denominators of the two probability density functions in Section 5.1.1. For $g_0(t)$, it should be t raised to the power of η_0+1 where the $+1$ was missing. A similar error appeared in $g_1(t)$, it should be t raised to the power of η_1+1.
- Version 1.00. Issued on 19 March 2024.

1 Introduction and learning objectives

You have learnt in Week 4A's lecture that a high variability of inter-arrival times or service times can cause a high response time. Measurements from real computer clusters have found that the service times in these clusters have very high variability [1]. The reference paper [1] also has a number of suggestions to deal with this issue. One suggestion is to separate the jobs according to their service time requirements, and have one set of servers processing jobs with short service times and another set of servers for jobs with long service times. This arrangement is the same as supermarkets having express checkouts for customers buying not more than a certain number of items and other checkouts that do not have a limit on the number of items. You had seen this theory in action in Week 4A's revision Problem 1. We also highly recommend you to read the paper [1].

In this project, you will use simulation to study how to reduce the response time of a server farm that uses different servers to process jobs with different service time requirements.

In this project, you will learn:

1. To use discrete event simulation to simulate a computer system
2. To use simulation to solve a design problem
3. To use statistically sound methods to analyse simulation outputs

We mentioned a number of times in the lectures that simulation is *not* simply about writing simulation programs. While it is important to get your simulation code correct, it is also important that you use statistically sound methods to analyse simulation outputs. There, roughly half of the marks of this project is allocated to the simulation program, and the other half to statistical analysis; see Section 7.2.

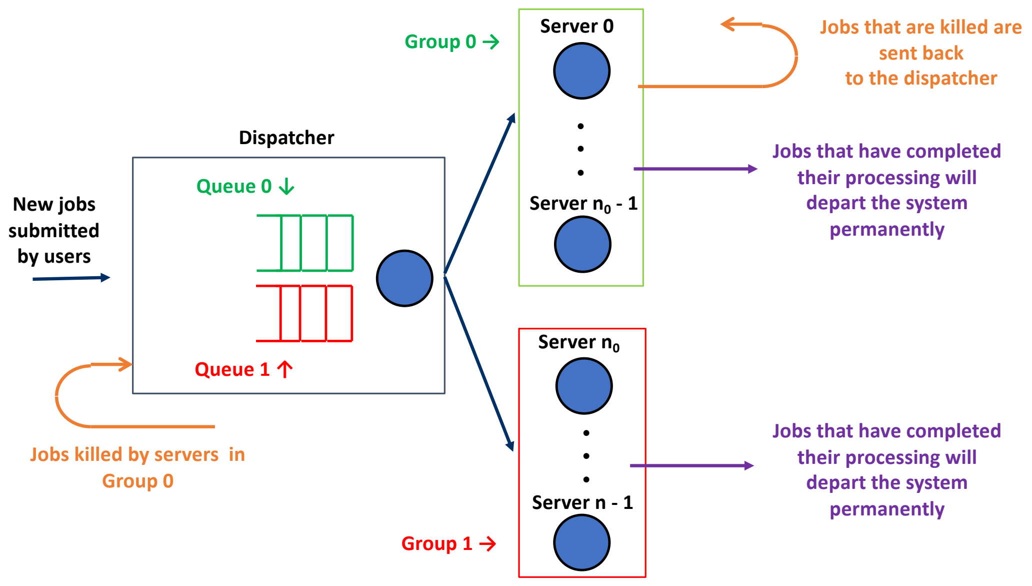
Group 0 →
Server 0
Jobs that are killed are sent back to the dispatcher
Dispatcher
Queue 0 ↓
New jobs submitted by users
Server n_0 - 1
Jobs that have completed their processing will depart the system permanently
Queue 1 ↑
Server n_0
Jobs killed by servers in Group 0
Server n - 1
Jobs that have completed their processing will depart the system permanently
Group 1 →

Figure 1: The multi-server system for this project.

2 Support provided and computing resources

If you have problems doing this project, you can post your question on the course forum. **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!** Please note that if your forum post shows part of your solution or code, you must mark that forum post **private**.

Another way to get help is to attend a consultation (see the Timetable section of the course website for dates and times).

If you need computing resources to run your simulation program, you can do it on the VLAB remote computing facility provided by the School. Information on VLAB is available here: `https://taggi.cse.unsw.edu.au/Vlab/`

3 Multi-server system configuration with job isolation

The configuration of the multi-server system that you will use in this project is shown in Figure 1. The system consists of a dispatcher and n servers where $n \geq 2$. The n servers are partitioned into 2 disjoint groups, called Groups 0 and 1, with at least one server in each group. The number of servers in Groups 0 and 1 are, respectively, n_0 and n_1 where $n_0, n_1 \geq 1$ and $n_0 + n_1 = n$.

The servers in Group 0 are used to process short jobs which require a processing time of no more than a time limit of T_{limit}. The servers in Group 1 do not impose any limit on service time.

The dispatcher has two queues: Queue 0 and Queue 1. The jobs in Queue i (where $i = 0, 1$) are destined for servers in Group i. Both queues have infinite queueing spaces.

When a user submits a job to this multi-server system, the user needs to indicate whether the job is intended for the servers in Group 0 or Group 1. The following *general processing steps* are common to all incoming jobs:

- If a job is intended for a server in Group i (where $i = 0, 1$) arrives at the dispatcher, the job will be sent to a server in Group i if one is available, otherwise the job will join Queue i.
- When a job departs from a server in Group i, the server will check whether there is a job at the head of Queue i. If yes, the job will be admitted to the available server for processing.

Recall that the servers in Group 0 have a service time limit. The intention is that the users make an estimate of the service time requirement of their submitted jobs. If a user thinks that their job should be able to complete within T_{limit}, then they submit it to Group 0; otherwise, they should send it to the Group 1.

Unfortunately, the service time estimated by the users is not always correct. It is possible that a user sends a job which cannot be completed within the time limit to Group 0. We will now explain how the multi-server system will process such a job. Since the user has indicated that the job is destined for Group 0, the job will be processed according to the general processing steps explained earlier. This means the job will receive processing by a server in Group 0. After this job has been processed for a time of T_{limit}, the server says that the service time limit is up and will *kill* the job. The server will send the job to the dispatcher and tell it that this is a killed job. The dispatcher will check whether a server in Group 1 is available. If yes, the job will be send to an available server; otherwise, it will join Queue 1 to wait for a server to become available. When a server in Group 1 is available to work on this job, it will process the job from the *beginning*, i.e., all the previous processing in a Group 0 server is lost.

If a job has completed its processing at a Group 0 server, which means its service time is less than or equal to T_{limit}, then the job leaves the multi-server system permanently. Similarly, a job completed its processing at a Group 1 server will leave the system permanently.

We make the following assumptions on the multi-server system in Figure 1. First, it takes the dispatcher negligible time to classify a job and to send a job to an available server. Second, it takes a negligible time for a server to send a killed job to the dispatcher. Third, it takes a negligible time for a server to inform the dispatcher on its availability. As a consequence of these assumptions, it means that: (1) If a job arriving at the dispatcher is to be sent to an available server right away, then its arrival time at the dispatcher is the same as its arrival time at the chosen server; (2) The departure time of a job from the dispatcher is the same as its arrival time at the chosen server; and (3) The departure time of a killed job from a server is the same as its arrival time at the dispatcher. Ultimately, these assumptions imply that the response time of the system depends only on the queues and the servers.

We have now completed our description of the operation of the system in Figure 1. We will provide a number of numerical examples to further explain its operation in Section 4.

You will see from the numerical examples in Section 4 that the number of Group 0 servers n_0 can be used to influence the mean response time. So, a design problem that you will consider in this project is to determine the value of n_0 to minimise the mean response time.

Remark 1 *Some elements in the above description are realistic but some are not. Typically, users are required to specify a `walltime` as a service time limit when they submit their jobs to a computing cluster. If a server has already spent the specified `walltime` on the job, then the server* *will kill the job. All these are realistic.*

The re-circulation of a killed job is normally not done. A user will typically have to resubmit a new job if it has been killed. If a killed job is re-circulated, then it may be given a lower priority, rather than joining the main queue which is the case here.

Some programming technique (e.g., checkpointing) allows a killed job or crashed job to resurrect from the last state saved rather than from the beginning. However, that may require a sizeable memory space.

In order to make this project more do-able, we have simplified many of the settings. For example, we do not use lower priority for the re-circulated killed jobs.

4 Examples

We will now present three examples to illustrate the operation of the system that you will simulate in this project. In all these examples, we assume that the system is initially empty.

4.1 Example 0: $n = 3$, $n_0 = 1$, $n_1 = 2$ and $T_{\text{limit}} = 3$

In this example, we assume the there are $n = 3$ servers in the farm with 1 ($= n_0$) server in Group 0 and 2 ($= n_1$) servers in Group 1. The time limit for Group 0 processing is $T_{\text{limit}} = 3$.

Table 1 shows the attributes of the 8 jobs that we will use in this example. Each job is given an index (from 0 to 7). For each job, Table 1 shows its arrival time, service time and the server group that the user has indicated. For example, Job 1 arrives at time 10, requires 4 units of time for service and the user has indicated that this job needs to go to a Group 0 server. Since the service time requirement for this job exceeds the time limit T_{limit} of 3, this job will be killed after 3 time units of service and will be sent to dispatcher after that.

Note that, a job which a user sends to a Group 0 server will be completed if its service time is *less than or equal to* the service time limit T_{limit} being imposed. So, Job 6 in Table 1 will be completed in a Group 0 server and this job will not be killed.

Job index	Arrival time	Service time required	Server group indicated
0	2	5	1
1	10	4	0
2	11	9	0
3	12	2	0
4	14	8	1
5	15	5	0
6	19	3	0
7	20	6	1

Table 1: Jobs for Example 0.

Remark 2 *We remark that the job indices are not necessary for carrying out the discrete event simulation. We have included the job index to make it easier to refer to a job in our description below.*

The events in the system in Figure 1 are

- The arrival of a new job to the dispatcher; and,

- The departure of a job from a server.

We remark that for a Group 1 server, a departed job has its service completed. However, for a Group 0 server, a departed job can be a killed job or a completed job. Note that we have not included the arrival of a *re-circulated killed* job to the dispatcher as an event. This is because the arrival of a re-circulated job at the dispatcher is *at the same time* as the departure of that job from a Group 0 server. So the simulation will handle these events together: the departure of a killed job and its handling by the dispatcher.

We will illustrate the simulation of the system in Figure 1 using "on-paper simulation". The quantities that you need to keep track of include:

- **Next arrival time** is the time that the next *new job* (i.e, not a killed job) will arrive
- For each server, we keep track its server status, which can be busy or idle.
- We also keep track of the following information on the job that is being processed in the server:
  - **Next departure time** is the time at which the job will depart from the server. If the server is idle, the next departure time is set to ∞. Note that there is a next departure time for each server.
  - The time that this job arrived at the system. This is needed for calculating the response time of the job when it permanently departs from the system.
- The contents of Queues 0 and 1. Each job in the queue is identified by a 2-tuple of (arrival time, service time).

There are other additional quantities that you will need to keep track of and they will be mentioned later on.

The "on-paper simulation" is shown in Table 2. The notes in the last column explain what updates you need to do for each event. Recall that the two event types in this simulation are the arrival of a *new job* to the dispatcher and the departure from a server, we will simply refer to these two events as *Arrival* and *Departure* in the "Event type" column (i.e., second column) in Table 2.

Master clock	Event type	Next arrival time	Server 0 Group 0	Server 1 Group 1	Server 2 Group 1	Queue 0	Queue 1	Notes
0	–	2	Idle, ∞	Idle, ∞	Idle, ∞	–	–	We assume the servers are idle and queues are empty at the start of the simulation. The next departure times for all servers are ∞. The "–" indicates that the queues are empty.
2	Arrival	10	Idle, ∞	Busy, (2,7)	Idle, ∞	–	–	This event is the arrival of Job 0 for a Group 1 server. Since both Group 1 servers are idle before this arrival, the job can be sent to any one of the idle servers. We have chosen to send this job to Server 1. The job requires a service time of 5, so its completion time is 7. Note that the record of the job in the server is a 2-tuple consisting of (arrival time, scheduled departure time). Lastly, we need to update the arrival time of the next job, which is 10.
7	Departure	10	Idle, ∞	Idle, ∞	Idle, ∞	–	–	This event is the departure of a job from Server 1. Since Queue 1 is empty, Server 1 becomes idle.
10	Arrival	11	Busy (10,13, 4)	Idle, ∞	Idle, ∞	–	–	This event is the arrival of Job 1 for a Group 0 server. Since Server 0 is idle, the job can be sent to the idle server. This job requires a service time of 4 which exceeds the service time limit of 3 for Group 0 servers, so the simulation needs to schedule this job to depart Server 0 at time 13 because this is the time that this job will be killed by the server. We use the 3-tuple consisting of (arrival time, scheduled departure time, service time), which for this job is (10, 13, 4), to indicate that this job arrives at time 10, is scheduled to depart at time 13 and its service time requirement is 4 time units. We need to include the service time of the job because we will need it later when the job is re-circulated to a Group 1 server. Note that if you see a 3-tuple job in a Group 0 server, it means that the job will be killed and re-circulated to a Group 1 server. Lastly, we need to update the arrival time of the next job, which is 11.

11	Arrival	12	Busy (10,13, 4)	Idle, ∞	Idle, ∞	(11,9)	–	This event is the arrival of Job 2 for a Group 0 server. Since Server 0 is busy, this job will join Queue 0. The queue stores the 2-tuple (arrival time, service time) which is (11,9) for this job. We also need to update the arrival time of the next job, which is 12.
12	Arrival	14	Busy (10,13, 4)	Idle, ∞	Idle, ∞	(11,9), (12,2)	–	This event is the arrival of Job 3 for a Group 0 server. Since Server 0 is busy, this job will join Queue 0 with the job information (12,2). We also need to update the arrival time of the next job, which is 14.
13	Departure	14	Busy (11,16, 9)	Busy (10,17)	Idle, ∞	(12,2)	–	This event is the departure of a killed job from Server 0. This job will be re-circulated to the dispatcher. Since both Group 1 servers are idle, this job can go to any one of them. We have chosen to send it to Server 1. Since this job requires 4 time units of service, it is scheduled to depart Server 1 at time 17. The 2-tuple (10,17) indicates that this job arrives at 10 and will depart at time 17. Since this is a departure from a Group 0 server, we will also need to check Queue 0, which has 2 jobs. So the job at the head of the queue will advance to Server 0 which is becoming available. This job requires 9 units of service time which exceeds the service time limit. So, the job will be killed at time 13 + 3 = 16 time units.
14	Arrival	15	Busy (11,16, 9)	Busy (10,17)	Busy (14,22)	(12,2)	–	This event is the arrival of Job 4 for a Group 1 server. Since there is a Group 1 server available, this job goes to Server 2 directly. This job requires 8 units of service, so the job is scheduled to depart at time 22. We also need to update the arrival time of the next job, which is 15.
15	Arrival	19	Busy (11,16, 9)	Busy (10,17)	Busy (14,22)	(12,2) (15,5)	–	This event is the arrival of Job 5 for a Group 0 server. Since all Group 0 servers are busy, this job joins Queue 0. We also need to update the arrival time of the next job, which is 19.

16	Departure	19	Busy (12,18)	Busy (10,17)	Busy (14,22)	(15,5)	(11, 9)	This event is the departure of a killed job from Server 0. This job will be re-circulated to the dispatcher. Since both Group 1 servers are busy, this job will join Queue 1. The job at the head of Queue 0 will advance to Server 0. This job requires only 2 units of service which is within the limit. We use a 2-tuple to remember this job because the job is within the time limit so it will not be killed.
17	Departure	19	Busy (12,18)	Busy (11, 26)	Busy (14,22)	(15,5)	-	This event is the departure of a finished job at Server 1. Since there is a job in Queue 1, the job will move into Server 1.
18	Departure	19	Busy (15,21,5)	Busy (11, 26)	Busy (14,22)	-	-	This event is the departure of a finished job at Server 0. This job will depart from the system permanently. We can tell that because it is a 2-tuple in the server rather than a 3-tuple. Since there is a job in Queue 0, the job will move into Server 0.
19	Arrival	20	Busy (15,21,5)	Busy (11, 26)	Busy (14,22)	(19,3)	-	This event is the arrival of Job 6 for a Group 0 server. Since all Group 0 servers are busy, this job joins Queue 0. We also need to update the arrival time of the next job, which is 20.
20	Arrival	∞	Busy (15,21,5)	Busy (11, 26)	Busy (14,22)	(19,3)	(20, 6)	This event is the arrival of Job 7 for a Group 1 server. Since all Group 1 servers are busy, this job joins Queue 1. Since there are no more jobs arriving, we update the next arrival time to ∞
21	Departure	∞	Busy (19,24)	Busy (11, 26)	Busy (14,22)	-	(20,6), (15,5)	This event is the departure of a killed job from Server 0. This job will be re-circulated to the dispatcher. Since both Group 1 servers are busy, this job will join Queue 1. The job at the head of Queue 0 will advance to Server 0. This job requires only 3 units of service which is within the limit. We only need a 2-tuple to remember that this job arrives at time 19 and will depart at time 24.
22	Departure	∞	Busy (19,24)	Busy (11, 26)	Busy (20, 28)	-	(15,5)	This event is the departure of a finished job at Server 2. Since there is a job in Queue 1, the job will move into Server 2.
24	Departure	∞	Idle, ∞	Busy (11, 26)	Busy (20, 28)	-	(15,5)	This event is the departure of a finished job at Server 0. Since Queue 0 is empty, Server 0 is now idle.
26	Departure	∞	Idle, ∞	Busy (15, 31)	Busy (20, 28)	-	-	This event is the departure of a finished job at Server 1. The job at the head of Queue 1 advances to Server 1. The queue is now empty.

28	Departure	∞	Idle, ∞	Busy (15, 31)	Idle, ∞	-	-	This event is the departure of a finished job at Server 2. Server 2 is now idle as Queue 1 is empty.
31	Departure	∞	Idle, ∞	Idle, ∞	Idle, ∞	-	-	This event is the departure of a finished job at Server 1. Server 1 is now idle as Queue 1 is empty.

Table 2: "On paper simulation" illustrating the event updates of the system.

The above description has not explained what happens if an arrival event and a departure event are at the same time. We will leave it unspecified. If we ask you to simulate in trace driven mode, we will ensure that such situation will not occur. If the inter-arrival time and service time are generated randomly, the chance of this situation occurring is practically zero so you do not have to worry about it.

Table 3 summarises the arrival, departure, job classification and response times of the jobs in this example. In the table, we classify the jobs into 3 types:

- Group 0 jobs that are completed (i.e., not killed) within the time limit. We will refer to these jobs as *completed Group 0 jobs* from now on. These jobs are marked as 0.
- Group 0 jobs that are recirculated. They are marked as r0.
- Jobs that are indicated for Group 1 by the users. They are marked as 1.

In Table 3, we have included the response times for completed Group 0 jobs and Group 1 jobs. The mean response time for completed Group 0 jobs is $\frac{11}{2} = 5.5$ and the mean response time for Group 1 jobs is $\frac{21}{3} = 7$.

Later on, you will work on a design problem to reduce a weighted sum of the mean response times of the completed Group 0 jobs and the Group 1 jobs. Here we have purposely neglected the re-circulated jobs because we will not attempt to reduce their response time. The reason is that we do not want to incentivise users to give poor estimation of the service time requirement of their jobs.

Job	Arrival time	Departure time	Job classification	Response time	
				Group 0 within limit	Group 1
0	2	7	1		5
1	10	17	r0		
2	11	26	r0		
3	12	18	0	6	
4	14	22	1		8
5	15	31	r0		
6	19	24	0	5	
7	20	28	1		8

Table 3: The arrival and departure times of the jobs in Example 0.

4.2 Example 1: $n = 4$, $n_0 = 2$, $n_1 = 2$ and $T_{\text{limit}} = 3.5$

For this example, we assume that the system has $n = 4$ servers. Both Groups 0 and 1 have 2 servers each, i.e., $n_0 = n_1 = 2$. The service time-limit for Group 0 server is $T_{\text{limit}} = 3.5$.

Table 4 shows the attributes of the jobs which will arrive at this system. Table 5 summaries the results of the simulation. The mean response time of the completed Group 0 jobs is $\frac{23.9}{4} = 5.975$ and the mean response time of the Group 1 jobs is $\frac{36.8}{5} = 7.36$.

Job index	Arrival time	Service time required	Server group indicated
0	2.1	5.2	1
1	3.4	4.1	1
2	4.1	3.1	0
3	4.4	3.9	0
4	4.5	3.4	0
5	4.7	4.4	1
6	5.5	4.7	1
7	5.9	4.1	0
8	6.0	2.5	0
9	6.5	8.6	1
10	7.6	4.1	0
11	8.1	2.6	0

Table 4: Jobs for Example 1.

Job	Arrival time	Departure time	Job classification	Response time	
				Group 0 within limit	Group 1
0	2.1	7.3	1		5.2
1	3.4	7.5	1		4.1
2	4.1	7.2	0	3.1	
3	4.4	16.1	r0		
4	4.5	10.6	0	6.1	
5	4.7	11.7	1		7.0
6	5.5	12.2	1		6.7
7	5.9	20.2	r0		
8	6.0	13.1	0	7.1	
9	6.5	20.3	1		13.8
10	7.6	24.3	r0		
11	8.1	15.7	0	7.6	

Table 5: The arrival and departure times of the jobs in Example 1.

4.3 Example 2: $n = 4$, $n_0 = 1$, $n_1 = 3$ and $T_{\text{limit}} = 3.5$

This example is identical to Example 1 except that $n_0 = 1$. Table 6 summaries the results of the simulation. The mean response time of the completed Group 0 jobs is $\frac{44.9}{4} = 11.225$ and the mean response time of the Group 1 jobs is $\frac{29.8}{5} = 5.96$. It is not surprising that the mean response time of the completed Group 0 jobs has gone up while that of Group 1 jobs has gone down. This is because in this example, there are fewer servers in Group 0.

Job	Arrival time	Departure time	Job classification	Response time	
				Group 0 within limit	Group 1
0	2.1	7.3	1		5.2
1	3.4	7.5	1		4.1
2	4.1	7.2	0	3.1	
3	4.4	14.6	r0		
4	4.5	14.1	0	9.6	
5	4.7	9.1	1		4.4
6	5.5	12.0	1		6.5
7	5.9	21.7	r0		
8	6.0	20.1	0	14.1	
9	6.5	16.1	1		9.6
10	7.6	27.7	r0		
11	8.1	26.2	0	18.1	

Table 6: The arrival and departure times of the jobs in Example 2.

5 Project description

This project consists of two main parts. The first part is to develop a simulation program for the system in Figure 1. The system has already been described in Section 3 and illustrated in Section 4. In the second part, you will use the simulation program that you have developed to solve a design problem.

5.1 Simulation program

You must write your simulation program in one (or a combination) of the following languages: Python 3 (note: version 3 only), C, C++, or Java. All these languages 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 and/or operating system 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.

Note that our description uses the following variable names:

1. A variable `mode` of string type. This variable is to control whether your program will run simulation using randomly generated arrival times and service times; or in trace driven mode. The value that the parameter `mode` can take is either `random` or `trace`.
2. A variable `time_end` which stops the simulation if the master clock exceeds this value. This variable is only relevant when `mode` is `random`. This variable is a positive floating point number.

Note that your simulation program must be a general program which allows different parameter values 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.

For the simulation, you can always assume that the system is empty initially.

Hint: Do **not** write two separate programs for the `random` and `trace` modes because they share a lot in common. A few `if–else` statements at the right places are what you need to have both modes in one program.

5.1.1 The random mode

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

1. We use $\{a_1, a_2, \dots, a_k, \dots, ...\}$ to denote the inter-arrival times of the jobs arriving at the dispatcher. These inter-arrival times have the following properties:
   (a) Each a_k is the product of two random numbers a_{1k} and a_{2k}, i.e $a_k = a_{1k}a_{2k}\ \forall k = 1, 2, ...$
   (b) The sequence a_{1k} is exponentially distributed with a mean arrival rate λ requests/s.
   (c) The sequence a_{2k} is uniformly distributed in the interval $[a_{2l}, a_{2u}]$.

   Note: The easiest way to generate the inter-arrival times is to multiply an exponentially distributed random number with the given rate and a uniformly distributed random number in the given range. It would be more difficult to use the inverse transform method in this case, though it is doable.

2. The workload of a job is characterised by two attributes: the server group (i.e., Group 0 or 1) that the job is to be sent to, and the service time of the job.

   (a) The first step to determine which server group to send the job to. This decision is made by a parameter $p_0 \in (0,1)$:
      - Prob[a job is indicated by the user for a Group 0 server] $= p_0$
      - Prob[a job is indicated by the user for a Group 1 server] $= 1 - p_0$

      For example, if p_0 is 0.8, then there is a probability of 0.8 that a job is indicated for a Group 0 server and a probability of 0.2 for a Group 1 server. The server group for each job is independently generated.

   (b) Once the server group for a job has been generated, the next step is to generate its service time. The service time distribution to be used depends on the server group.

      i. If a job is indicated to go to a Group 0 server, its service time has the probability density function (PDF) $g_0(t)$:

$$g_0(t) = \begin{cases} 0 & \text{for } 0 \le t \le \alpha_0 \\ \dfrac{\eta_0}{\gamma_0 \; t^{\eta_0+1}} & \text{for } \alpha_0 < t < \beta_0 \\ 0 & \text{for } t \ge \beta_0 \end{cases} \tag{1}$$

      where

$$\gamma_0 = \alpha_0^{-\eta_0} - \beta_0^{-\eta_0}$$

      Note that this probability density function has 3 parameters: α_0, β_0 and η_0. You can assume that $\beta_0 > \alpha_0 > 0$ and $\eta_0 > 1$.

      ii. If a job is indicated to go to a Group 1 server, its service time has PDF:

$$g_1(t) = \begin{cases} 0 & \text{for } 0 \le t \le \alpha_1 \\ \dfrac{\eta_1}{\gamma_1 \; t^{\eta_1+1}} & \text{for } \alpha_1 < t \end{cases} \tag{2}$$

      where

$$\gamma_1 = \alpha_1^{-\eta_1}$$

      Note that this probability density function has 2 parameters: α_1 and η_1. You can assume that $\alpha_1 > 0$ and $\eta_1 > 1$.

5.1.2 The trace mode

When your simulation is working in the `trace` mode, it will read the list of **inter-arrival** times, the list of service times and server groups from two separate ASCII files. We will explain the format of these files in Sections 6.1.3 and 6.1.4.

An **important requirement** for the `trace` mode is that your program is required to simulate until all jobs have departed from the system. You can refer to Table 2 for an illustration.

5.2 Determining the value of n_0 that minimises a weighted mean response time

After writing your simulation program, your next step is to use your simulation program to determine the number of Group 0 servers n_0 that minimises a weighted mean response time.

For this design problem, you will assume the following parameter values:

- Total number of servers: $n = 10$
- The service time limit T_{limit} for Group 0 servers is 3.3.
- For inter-arrival times: $\lambda = 3.1$, $a_{2\ell} = 0.85$, $a_{2u} = 1.21$
- The probability p_0 that a job is indicated for a Group 0 server is 0.74.
- The service time for a job which is indicated for Group 0: $\alpha_0 = 0.5$, $\beta_0 = 5.7$, $\eta_0 = 1.9$.
- The service time for a job which is indicated for Group 1: $\alpha_1 = 2.7$ and $\eta_1 = 2.5$.

The aim of the design problem is to minimise the weighted response time:

$$w_0 T_0 + w_1 T_1 \tag{3}$$

where T_0 is the mean response time of the completed Group 0 jobs and T_1 is the mean response time of Group 1 jobs. The value of the weights w_0 and w_1 are fixed for this design problem, and they are given by 0.83 and 0.059 respectively. As an example, if $T_0 = 1.86$ and $T_1 = 56.7$, then the weighted mean response time is $0.83 \times 1.86 + 0.059 \times 56.7$. The rationale behind choosing these weights is explained in Remark 3.

The aim of the design problem is to find the value of n_0 to minimise this weighted response time. Note that we assume that there is at least a server in each group, therefore $1 \leq n_0 \leq n - 1$.

In solving this design problem, you need to ensure that you use **statistically sound** methods to compare systems. You will need to consider simulation controls such as length of simulation, number of replications, transient removals and so on. You will need to justify in your report on how you determine the value of n_0.

Remark 3 *For the parameters above, out of all the jobs that are not re-circulated, 73.65% are Group 0 jobs within the time limit and 26.35% are Group 1 jobs. The average service time for Group 0 jobs within the time limit is 0.887 and that for Group 1 jobs is 4.5. The weights w_0 and w_1 are computed, respectively, from $\frac{0.7365}{0.887}$ and $\frac{0.2635}{4.5}$. So the weights take into account the frequency of a class of jobs. We also use the inverse service time as a weight so that we are not giving too much advantage to Class 1 jobs as they have large service time requirement.*

6 Testing your simulation program

In order for us to test the correctness of your simulation program, we will run your program using a number of test cases. The aim of this section is to describe the expected input/output file format and how the testing will be performed.

Each test is specified by 4 configurations files. We will index the tests from 0. If 12 tests are used, then the indices for the tests are 0, 1, 2, ..., 11. The names of the configuration files are:

- For Test 0, the configuration files are `mode_0.txt`, `para_0.txt`, `interarrival_0.txt` and `service_0.txt`. The files are similarly named for indices 1, 2, 3, .., 9.
- For Test 10, the configuration files are `mode_10.txt`, `para_10.txt`, `interarrival_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. We will describe the format of the configuration files in Section 6.1

Each test should produce 2 output files whose format will be described in Section 6.2. We will explain how testing will be conducted in Sections 6.3 and 6.5.

6.1 Configuration file format

Note that Test 0 is the same as Example 0 discussed in Section 4.1. We will use that test to illustrate the file format.

6.1.1 `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`.

6.1.2 `para_*.txt`

If the simulation mode is `trace`, then this file has three lines. The first line is the value of n (= total number of servers), the second line has the value of n_0 (= number of Group 0 servers) and the third line has the value of T_{limit}. If the test is Example 0 in Section 4.1, then the contents of this file are:

```
3
1
3
```

These values are in the sample file `para_0.txt`.

If the simulation mode is `random`, then the file has four lines. The meaning of the first three lines is the same as above. The last line contains the value of `time_end`, which is the end time of the simulation. The contents of the sample file `para_4.txt` are shown below where the last line indicates that the simulation should run until `200`.

```
5
2
3.1
200
```

You can assume that we will only give you valid values. You can expect n to be a positive integer greater than 2, $n_0 \geq 1$ and $T_{\text{limit}} > 0$. For `time_end`, it is a strictly positive integer or floating point number.

6.1.3 `interarrival_*.txt`

The contents of the file `interarrival_*.txt` depend on the `mode` of the test. If mode is `trace`, then the file `interarrival_*.txt` contains the interarrival times of the jobs with one interarrival time occupying one line. You can assume that the list of interarrival times is always positive. For Example 0 in Section 4.1, the arrival times are $[2, 10, 11, 12, 14, 15, 19, 20]$ which means the inter-arrival times are $[2, 8, 1, 1, 2, 1, 4, 1]$. For this example, the inter-arrival times will be specified by a file (see sample file `interarrival_0.txt`) whose contents are:

```
2.0000
8.0000
1.0000
1.0000
2.0000
1.0000
4.0000
1.0000
```

If the mode is `random`, then the file `interarrival_*.txt` contain three numbers in one line. These three numbers correspond to the parameters λ, $a_{2\ell}$ and a_{2u}. As an example, the contents of `interarrival_4.txt` are:

```
0.9 0.91 1.27
```

For this example, the values of λ, $a_{2\ell}$ and a_{2u} are respectively 0.9, 0.91 and 1.27. You can assume that all these parameter values are positive.

6.1.4 `service_*.txt`

For `trace` mode, the file `service_*.txt` contains, for each job, its service time and the server group for which the job is destined. As an illustration, the service times and server groups for Example 0 in Section 4.1 will be specified by a file (see sample file `service_0.txt`) whose contents are:

```
5.0000 1
4.0000 0
9.0000 0
2.0000 0
8.0000 1
5.0000 0
3.0000 0
6.0000 1
```

Note that each row has 2 entries, and they correspond to the service time (first entry) and the server group (second entry). For example, the first job has a service time of 5 and is indicated for a Group 1 server. You will find a one-to-one correspondence between the content of `service_0.txt` and the information in Table 1. You can assume that the first entry is a positive float, and the second entry in each row is either 0 or 1.

For `random` mode, the file `service_*.txt` contains three lines. For example, the contents of `service_4.txt` are:

```
0.7
1.2 3.6 2.1
2.8 4.1
```

The number in the first line is p_0. The three numbers in the second line are α_0, β_0 and η_0. Finally, the two numbers in the third line are α_1 and η_1. You can assume all these values are valid.

You can assume that the data we provide for `trace` mode are consistent in the following way: the number of inter-arrival times and the number of lines of service times are equal.

6.2 Output file format

In order to test your simulation program, we need two output files **per test**. One file contains two mean response times. The other file contains the arrival times, departure times and job classification information similar to Columns 2–4 in Table 3.

For `random` mode, the mean response time should be calculated using those jobs that have permanently departed the system by `time_end`. In other words, for those jobs which are still in the queue or are being processed in the server at `time_end`, you do not include these jobs when calculating the mean response time.

Note that you do not have to consider transient removal for the mean response before you write the result to the output file. However, you should consider transient removal when you do your design.

Two mean response times should be written to a file whose filename has the form `mrt_*.txt`. For Example 0 in Section 4.1, the expected contents of this file are:

```
5.5000 7.0000
```

where the two numbers correspond to the mean response times of, respectively, the completed Class 0 and Class 1 jobs.

The other file `dep_*.txt` contains the departure type and classification of the jobs. For Example 1 in Section 4.2, the expected contents of this file are:

```
4.1000 7.2000 0
2.1000 7.3000 1
3.4000 7.5000 1
4.5000 10.6000 0
4.7000 11.7000 1
5.5000 12.2000 1
6.0000 13.1000 0
8.1000 15.7000 0
4.4000 16.1000 r0
5.9000 20.2000 r0
6.5000 20.3000 1
7.6000 24.3000 r0
```

Note the following requirements for the file:

1. Each line contains 3 entries.
2. For each line, the first entry is the arrival time of the job to the system (i.e., as a new job), the second entry is its permanent departure time from the system and the third entry is a classification of the job in the same way as Column 4 in Table 3. The possible classifications for a job are `0`, `r0` and `1`. You should be able to reconcile the contents of the above file with Example 1 in Section 4.2.
3. The jobs must be ordered according to *ascending* completion times.
4. If the simulation is in the `trace` mode, we expect the simulation to finish after all jobs have been processed. Therefore, the number of lines in `dep_*.txt` should be equal to the number of jobs.
5. If the simulation is in the `random` mode, the file should contain all the jobs that have been completed by `time_end`.

All mean response times, arrival times and completion times in `mrt_*.txt` and `dep_*.txt` should be printed as floating point numbers to exactly 4 decimal places. Note that your simulation should be performed in full floating point precision and you should only do the rounding when you are writing the output files.

6.3 The testing framework

When you submit your project, you must include a Linux bash shell script with the name `run_test.sh` so that we can run your program on the CSE system. This shell script is required because you are allowed to use a computer language of your choice.

Let us first recall that each test is specified by four configuration files and should produce two output files. For example, test number 0 is specified by the configuration files `mode_0.txt`, `interarrival_0.txt`, `service_0.txt` and `para_0.txt`; and test number 0 is expected to produce the output files `mrt_0.txt` and `dep_0.txt`.

We will use the following directory structure when we do testing.

```
the directory containing run_test.sh
 ├── config/
 └── output/
```

We will put all the configuration files for all the tests in the sub-directory `config/`. You should write all the output files to the sub-directory `output/`.

To run test number 0, we use the shell command:

```
./run_test.sh 0
```

The expected behaviour is that your simulation program will read in the configuration files for test number 0 from `config/`, carry out the simulation and create the output files in `output/`.

Similarly, to run test number 1, we use the shell command:

```
./run_test.sh 1
```

This means that the shell script `run_test.sh` has one input argument which is the test number to be used.

Let us for the time being assume that you use Python (Version 3) to write your simulation program and you call your simulation program `main.py`. If the file `main.py` is in the same directory as `run_test.sh`, then `run_test.sh` can be the following one-line shell script:

```
python3 main.py $1
```

The shell script will pass the test number (which is in the input argument `$1`) to your simulation program `main.py`. This also implies that your simulation program should accept one input argument which is the test number.

Just in case you are not familiar with shell script, we have provided two sample files: `run_test.sh` and `main.py` to illustrate the interaction between a shell script and a Python (Version 3) file. You need to make sure `run_test.sh` is executable. (You can make the shell script `run_test.sh` executable by using the command "`chmod u+x run_test.sh`".) If you run the command `./run_test.sh 2`, it will produce a file with the name `dummy_2.txt` in the directory `output/`. You can also try using other input arguments for the sample shell script. You can use these sample files to help you to develop your code.

If you use C, C++ or Java, then your `run_test.sh` should first compile the source code and then run the executable. You should of course pass the test number to the executable as an input.

You can put your code in the same directory that contains `run_test.sh` or in a subdirectory below it. For example, you may have a subdirectory `src/` for your code like the following:

```
the directory containing run_test.sh
 ├── config/
 ├── output/
 └── src/
```

6.4 Sample files

You should download the file `sample_project_files.zip` from the project page on the course website. The zip archive has the following directory structure:

```
Base directory containing cf_output_with_ref.py, run_test.sh and main.py
├── config/
├── output/
└── ref/
```

Details on the zip-archive are:

- The sub-directory `config/` contains configuration files that you can use for testing.
  - The files `mode_0.txt`, `mode_1.txt`, ..., and `mode_7.txt`. Note that Tests 0–3 are for `trace` mode while Tests 4–6 are for `random` mode.
  - The files `para_*.txt`, `interarrival_*.txt` and `service_*.txt` for `*` from 0 to 6, as the input to the simulation.
  - Note that Tests 0–2 are the same as Examples 0–2 in Section 4.
- The sub-directory `output/` is empty. Your simulation program should place the output files in this sub-dirrectory.
- The sub-directory `ref/` contains the expected simulation results.
  - The files `mrt_*_ref.txt` and `dep_*_ref.txt` for `*` from 0 to 6, as the reference files for the output. For Tests 0–3, you should be able to reproduce the results in `mrt_*_ref.txt` and `dep_*_ref.txt`. However, since Tests 4–6 are in `random` mode, you will **not** be able to reproduce the results in the output files. They have been provided so that you can check the expected format of the files.
- The Python file `cf_output_with_ref.py` which illustrates how we will compare your output against the reference output. This file takes in one input argument, which is the test number. For example, if you want to check your simulation outputs for test 0, you use:

  ```
  python3 cf_output_with_ref.py 0
  ```

  Note the following:
  - The file `cf_output_with_ref.py` expects the directory structure shown earlier.
  - For `trace` mode, we will check your mean response times, the departure times and classifications. Note that we are not looking for an exact match but rather whether your results are within a valid tolerance. The tolerance for the `trace` mode is 10^{-3} which is fairly generous for numbers with 4 decimal places.
  - For `random` mode, we will only check the mean response times. You can see from the sample file that we check whether the mean response time is within an interval. We obtain this interval using the following method: (i) we first simulate the system many times; (ii) we then use the simulation results to estimate the maximum and minimum mean response times; (iii) we use the estimated maximum and minimum values to form an interval; (iv) in order to provide some tolerance due to randomness, we enlarge this interval further.
  - Note that we use a very generous tolerance so if your mean response time does not pass the test, then it is highly likely that your simulation program is not correct.
- The files `run_test.sh` and `main.py` as mentioned in Section 6.3.

6.5 Carrying out your own testing on the CSE system

It is important for you to note the assumption on directory structure mentioned in Section 6.3. You must ensure your shell script and program files are written with this assumption in mind.

Since we will be testing your work on the CSE system, we strongly advise you to carry out the following on the CSE system before submission.

- Create a new folder in your CSE account and `cd` to that folder. We will refer to this directory as the base directory.
  - Copy your shell script `run_test.sh` and program files to the base directory
  - Copy the `config` and `ref` directories, as well as their contents, to the base directory
  - Create an empty directory `output`
- Make sure your shell script is executable by using the command "`chmod u+x run_test.sh`"
- Run your shell script for each test one by one. Make sure that each run produces the appropriate output files for that test in the `output` directory.
- Copy `cf_output_with_ref.py` to the base directory. Run it to compare your output against the reference output.

These steps are the same as those that we will use for testing. It is important to know that we will create an empty `output/` directory before we run your code. This means your code does **NOT** have to create the `output/` directory.

The submission portal will make an attempt to run test number 0 with your submitted files, see Section 7.3.

6.6 Getting started and base code

For this project, we do not require you to write your code from scratch. You are allowed to build your project by using: (i) the sample code from COMP9334; or (ii) the code in the public domain as long as it meets the requirements below.

If you intend to use Python 3 to write your simulation code, the best way to get started is to use the M/M/m simulation code provided with the solution to Week 4B's revision problem and modify from there. Sample code for trace driven simulation is provided with the lecture in Week 4B.

There is also a lot of discrete event simulation code in Python 3, C, C++ and Java in the public domain. You are allowed to use the public domain code as a basis for your project work as long as it meets the following requirements:

1. The code has a clearly identifiable author
2. The code has a date which is before the date that this project document is released.
3. You provide us with an URL of the source code.
4. You clearly state the changes that you have made on the original code to adapt it to the specifications of this project.

If you use any public domain code in your project, your project report **must** include the information to satisfy the above four requirements.

If you would like to use a certain public domain source but you are not sure whether it meets our requirements, you can consult the lecturer on the forum using a private message.

If your project work is based on the COMP9334 sample code, then your report **must** state that the COMP9334 sample code has been used and provide information to satisfy Requirement 4 above.

7 Project requirements

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

7.1 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".
   - (d) The report must include the information required in Section 6.6.
2. Program source code:
   - (a) For doing simulation
   - (b) The shell script `run_test.sh`, see Section 6.3.
3. 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 running your code on the CSE system. It is important that you submit the right version of the code and make sure that it runs on the CSE system.

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 7.2 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.

7.2 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
   - (b) Look for evidence in your report that you have verified the correctness of the inter-arrival probability distribution, the probability of sending a job to Group 0 or Group 1, and service time distribution. 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. Although we have given you test cases, we have *at no point* claimed that those test cases are sufficient to verify the correctness of your simulation output.

   You can meet this assessment criterion by arguing that the test cases that we have provided are sufficient and why. Alternative. you may derive test cases to test your code and explain the rationale of your new test cases. You can include appropriate supporting materials to demonstrate this in your submission.

2. You will need to demonstrate that your results are reproducible. You should provide evidence of this in your report.

3. For the part on determining a suitable value of n_0 that minimises the weighted mean response time, 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.

The above marking criteria closely follow the messages that we have been promoting in our lectures on discrete event simulation. You need to ensure that your simulation code is correct and at the same time you need to consider the choice of simulation parameters and use statistical sound method to compare systems. If you want to do well for the project, you must make sure that you cover all the above aspects. *Roughly half of the project marks go to Points 1 and 2 above, and roughly half of the project marks go to Point 3 above.*

7.3 How to submit

You should "zip" your report, shell script, programs and supporting materials into a file called "project.zip". The submission system will only accept this filename. If you need to store directories when zipping, you need to use the `-r` switch to preserve the relative path.

You should submit your work via the course website. Your submission cannot be more than 20MBytes in size.

You can submit multiple times before the deadline. A later submission overrides the earlier submissions, so make sure you submit the correct file. We will only mark the last submission that you make. 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.

When you submit your files, the submission portal will unzip your `project.zip` and run a test script. The script will search for your `run_test.sh` (using the shell command `find . -name test.sh`) and executes sample test 0 if a unique `run_test.sh` is find. If the test script says that it cannot find your `run_test.sh` or it finds multiple files with the name `run_test.sh`, then you should resubmit and you should ensure that there is exactly one `run_test.sh` file in your zip archive. You can do this test after you have got the simulation part ready and before you attempt the design. Since later submissions will overwrite the earlier ones, you can get this test done earlier.

8 Further project conditions

1. The total mark for this project is 30 marks.

2. The submission deadline is 5:00pm Friday 19 April 2024. Submissions made after the deadline will incur a penalty of 5% per day. The penalty is applied to the mark that you would have received if the submission was not late. Late submissions will only be accepted until 5:00pm Wednesday 24 April 2024, after which no submissions will be accepted.

3. If you use a computer program to perform any part of your work, you **must** submit the program or you lose marks for that component. This requirement applies to computer programs for simulation as well as those for statistical analysis.

4. Additional project conditions:

   - Joint work is not permitted on this project.
     - This is an individual project. As stated in Section 6.6, you must identify the source of the code that you have used, whether it comes from COMP9334 or public domain.
     - Do not request help from anyone other than the teaching staff of COMP9344.
     - Do not post your project work or code to the course forum.
     - project submissions are routinely examined both automatically and manually for work written by others.

     Rationale: this project is designed to develop the individual skills needed to solve problems. Using work/code written by, or taken from, other people will stop you learning these skills. Other CSE courses focus on skills needed for working in a team.
   - The use of AI generative tools, such as ChatGPT, is not permitted on this project.

     Rationale: We have given you the permission to use public domain code as a basis to develop your project, so it is not necessary for you to use ChatGPT. Our test with ChatGPT found that it was not able to supply us with a piece of complete running code for simulating a M/M/1 queue.
   - Sharing, publishing, or distributing your project work is not permitted.
     - Do not provide or show your project work to any other person, other than the teaching staff of COMP9334. For example, do not message your work to friends.
     - Do not publish your project code via the Internet. For example, do not place your project in a public GitHub repository.

     Rationale: by publishing or sharing your work, you are facilitating other students using your work. If other students find your project work and submit part or all of it as their own work, you may become involved in an academic integrity investigation.
   - Sharing, publishing, or distributing your project work after the completion of COMP9334 is not permitted.
     - For example, do not place your project in a public GitHub repository after this offering of COMP9334 is over.

     Rationale: COMP9334 may reuse project themes covering similar concepts and content. If students in future terms find your project work and submit part or all of it as their own work, you may become involved in an academic integrity investigation.

References

[1] Mor Harchol-Balter and Ziv Scully. The Most Common Queueing Theory Questions Asked by Computer Systems Practitioners.*First International Workshop on Teaching Performance Analysis of ComputerSystems (TeaPACS 2021) In conjunction with the IFIP Performance 2021 Conference.* Milan, Italy, Nov 2021. DOI 10.1145/3543146.3543148

[2] Mor Harchol-Balter. Performance Modeling and Design of Computer Systems. Cambridge University Press (2013).