Programming Project 2023-24

Version 3 - 4/12/2023

The project requires to develop a Java program that simulates the scheduling of jobs in a multiserver system. The system to be simulated consists of K identical *servers*, denoted by S_i, for $0 \leq i < K$, which are available to execute *jobs* of H *categories*, denoted by C_r, for $0 \leq r < H$. The arrival and service times of the jobs of each category are generated at random according to specific distributions. In particular, for each category C_r:

- The *interarrival time* between two successive jobs of this category is a random value with *exponential distribution*[1] of parameter $\lambda_r^{\text{arrival}} > 0$. Note that each interarrival time is a newly generated random value.
- The *service time* required by a job of this category is a random value with *exponential distribution* of parameter $\lambda_r^{\text{service}} > 0$. Note that for every job, its service time is a newly generated random value.

It is known that the mean of the exponential distribution of parameter λ is $1/\lambda$ (i.e., if you generate many independent random values with that distribution, their arithmetic mean tends to $1/\lambda$).

When a new job J arrives it is assigned to a server S according to some *scheduling policy* and, if S is busy serving some other job, J is put in the FIFO queue associated with S and will be scheduled for execution in S as soon as the job currently executed by S, and all other jobs preceding J in the queue, have finished their execution.

In order to simulate the above system you will employ the Discrete Event Simulation methodology [1, 2]. Based on this methodology, you will simulate only the following **two types of events**: (1) *arrival of a new job* J; and (2) *end of execution of a job* J *at a server* S. Observe that these are the only two events that modify the configuration of the jobs in the system. Each event e will be represented by an *entry*

$$e = (t_e, v_e),$$

where the key t_e is the time when e occurs, while the value v_e carries the relevant information about the event.

The purpose of the simulation is to gather the following statistics regarding the set A of the first N jobs arrived.

- End Time (**ET(A)**) when the simulation ends, that is the latest time when a job of A ends its execution.

[1] Details on the exponential distribution can be found in Wikipedia (`en.wikipedia.org/wiki/Exponential_distribution`). However, for the project you must only know how to generate random values with this distribution, and this will be explained later.

- The Average Queueing Time of all jobs (**AQT-all(A)**), defined as

$$\sum_{J \in A} q_J / N,$$

  where $q_J = t_J(2) - t_J(1)$, with $t_J(2)$ being the time when J started its execution at a server, and $t_J(1)$ being the time when J arrived.

- For $0 \leq r < H$, the Average Queueing Time of jobs of category C_r (**AQT(A,r)**), defined as

$$\sum_{J \in A_r} q_J / N_r,$$

  where A_r is the subset of jobs in A of category C_r, $N_r = |A_r|$, and q_J is as defined above.

1 Structure of the simulation

The events are simulated in order of occurrence, i.e., in non-decreasing order of key. To this purpose, a Priority Queue Q is employed, and the skeleton of the simulation is as follows.

- First, Q is initialized with the H events representing the arrivals of the first jobs of the various categories, where the arrival time of the first job of category C_r is a random value with exponential distribution of parameter $\lambda_r^{\text{arrival}}$, for $0 \leq r < H$.

- Then, a loop is executed where in each iteration an entry $e = (t_e, v_e)$ with minimum key is extracted from Q and processed, depending on the type of event which e represents:

  - **Arrival of a job J of category C_r.** The following operations are performed:
    1. A new entry representing the arrival of the next job of category C_r is added to Q (the arrival of this new job will be at time $t_e + X$, where X is a random value with exponential distribution of parameter $\lambda_r^{\text{arrival}}$);
    2. A server S is selected for J according to a scheduling policy and, if S is busy, J is put in the FIFO queue of S, otherwise J is scheduled for immediate execution in S. In this latter case (J immediately executed) a new entry is added to Q, which represents the end of the execution of J at time $t_e + s_e$, where s_e is the *service time* of the job J; s_e is computed as a random value with exponential distribution of parameter $\lambda_r^{\text{service}}$.

  - **End of execution of a job J of category C_r at server S.** If the FIFO queue of S is not empty, the first job J' is removed from this queue and scheduled for immediate execution in S, inserting in Q a new entry which represents the end of the execution of J'. In this case, the time when J' ends its execution is obtained as explained above, based on the category of J'.

Clearly, after processing each entry, all variables which are needed to compute the final statistics must be suitably updated.

2 Generation of random values with exponential distribution

As explained in [2, §3.2], a random value X with exponential distribution of parameter λ is generated as

$$X = -\frac{1}{\lambda}\ln(1-\alpha),$$

where α is a random value in $[0,1)$ with uniform distribution. The value α can be generated using method `nextFloat` from an instance of class `Random` (package `java.util`). You will have to use $2H$ distinct generators for the interarrival and service times of the H categories. Each generator will be created by invoking `new Random(seed)`, where `seed` is a suitable seed for the generator that is given in input.

3 Assignment

You must develop a Java program `Simulator.java` which reads the input parameters from a text file whose path is given as a command-line argument. The file contains the following data:

- Line 1 contains the following parameters separated by comma: K (number of servers), H (number of categories), N (total number of jobs to be simulated), R (repetitions of the simulation), and P (the type of scheduling policy to use).
- Line $2+r$, for $0 \le r < H$, contains the following parameters separated by comma: $\lambda_r^{\text{arrival}}$, $\lambda_r^{\text{service}}$ (parameters of the interarrival and service time distributions), and $\texttt{seed}_{r,arr}$, $\texttt{seed}_{r,ser}$, the seeds to be used, respectively, for the random generators of the interarrival and service times of category C_r.

After reading the input parameters from the file, the program simulates the first N jobs that arrive in the system (set A). Any job arriving after the N-th one must be ignored, even if it arrives when some job of A has not yet completed its execution. The simulation follows the structure described in Section 1.

The parameter P denotes the type of policy that the algorithm should use. The standard scheduling policy ($P = 0$) selects a server S for the i-th newly arrived job J, with $0 \le i \le N-1$, following a round-robin scheme: *allocate the job J to the server S_x with index $x = i \bmod K$*. The program may also implement a custom, alternative scheduling policy (that is activated with $P = 1$); you may design this policy freely. Note that it is not mandatory to implement a custom policy, but if you do so you may receive a higher score (see Section 4 for more details on grading).

The program must run the simulation R times (but the generators must be initialized only once), and based on the data collected in the R runs, the following information must be printed in the standard output in each line:

- K, H, N, R, P
- (only if $R = 1$, $N \leq 10$, and $P = 0$) the $2N$ events extracted from Q. For each event $e = (t_e, v_e)$, relative to some job J, you must print the values t_e, s_e, c_e, where: t_e is the time of occurence, s_e is the service time of J, if the event e is the end of execution of J, and 0 otherwise; and c_e is the category of J. Each of the $2N$ events must be printed in a different line, in order of time t_e.
- End Time ET(A). The value to be printed is the average of the End Times over the R runs.
- Average Queuing Time AQT-all(A) (average over the R the runs).
- For each category C_r, with $0 \leq r < H$:
  - number N_r of simulated jobs of C_r,
  - Average Queueing Time AQT(A,r),
  - Average service time of the simulated jobs of C_r, i.e., the arithmetic mean of their service times

  The printed values must be averages over R runs, and must be printed using one line per category.

Do not print other data or text in the output other than what is described above. We will provide you examples of input and output files for several parameter settings on the Moodle Exam webpage.

4 Grading

The grading scheme will be the following.

- **2 points:** if the program is correct and reasonably efficient for large N and K.
- **1 extra point:** if a good custom scheduling policy ($P = 1$) is implemented efficiently. The extra point will be assigned if the new policy reduces ET(A) compared to the standard policy ($P = 0$), while keeping the program correct and efficient for large N and K. A brief high-level description of the new policy and of its implementation must be included as a comment at the beginning of the program.

5 Deliverables

Submit the `Simulator.java` to the dedicated section in the Moodle Exam site. The program cannot call library functions other than those in java.lang, java.io, and java.util.

IMPORTANT: do not change the name of the Java file, and make sure that your program compiles and runs (with input and output formatted exactly as described before) using the commands `javac Simulator.java` and `java Simulator parameters.txt`. Note that we will run your implementations with automated scripts to parse the output of your programs; **if the format of the output is not correct, we will not be able to evaluate your submission.**

References

[1] Wikipedia: `en.wikipedia.org/wiki/Discrete-event_simulation`

[2] G. Iazeolla. *Introduzione alla simulazione discreta*, Boringhieri, 1978