Experimental Methods in Computer Science

2020/2021

This assignment is a first (and simple) exercise of designing experiments. There are many types of experiments and the term “design of experiments” (often referred to as experimental design, as well) is used for the process of defining and planning experiments in such a way that the data obtained can be analysed to draw valid and objective conclusions.

The topic of designing experiments of different types and for different purposes is addressed in detail in the Lectures. In any case, the next paragraphs offer a brief overview of this topic before presenting the assignment itself.

A classic experiment includes the following elements (you can take them as basic steps for the purpose of this assignment):

1. Problem statement (or research question)
2. Identify variables
3. Formulate hypothesis
4. Define the experimental setup/scenario
5. Develop the necessary tools and procedures for the experiment
6. Run the experiments and collect the data/measurements
7. Perform data analysis
8. Draw conclusions (and often go back to the beginning and reformulate the problem statement or test a different hypothesis)

The formulation of sound problem statements (or research questions) is not easy, and this is particularly true for the computer science field. A good (i.e., relevant) problem statement should be focused enough to allow the variables of the problem to be clearly identified, but, at the same time, it should be sufficiently open to allow different hypothesis to be formulated.

An example of formulation of a possible problem statement is the following:

How does the setting (noise, temperature, music, open space, etc.) of the room used by programmers affect the number of bugs found by test suites in the modules produced by those programmers? The effect we want to measure (dependent variable) is the number of bugs found and the factors (independent variables) are the different room settings and situations. It is assumed that there is a set of conditions that remain stable (e.g., type of programming language, complexity of the modules, etc.).

A possible hypothesis for the problem statement above is that programmers tend to introduce fewer bugs if they listen to classical music at a comfortable volume while programming. In this case, the

hypothesis is directional (i.e., it establishes a direction for the dependency between the dependent variable and the independent variable) but the hypothesis could simply state that music influences the number of bugs (non-directional hypothesis).

Even in a simple example like this, it is easy to understand the difficulties associated with the correct validation of hypotheses. In the current example, it would be necessary to perform experiments with a representative group of programmers, use a variety of software under development, ensure that the conditions of the different experimental runs remain stable, etc. The analysis of the results in order to check whether they support the hypothesis is often quite difficult as well, often requiring complex statistical testing techniques.

1 Goals, assumptions and recommendations

The purpose of this assignment is to design a set of experiments, perform them, and analyse the resulting data in order to find answers to the general problem statement:

> How do process scheduling algorithms affect the performance of computer systems under different workloads?

Modern operating systems try to take as much advantage of a computer's Central Processing Unit (CPU) as possible by allowing it to be shared by a number of software processes. The execution of a typical process consists of a sequence of so-called CPU bursts interleaved with Input/Output (I/O) bursts. In other words, the CPU is assigned to a process at the beginning of its execution, but at some point the process may have to wait until an I/O request is fulfilled before it can continue to run. In that case, giving the CPU to another process that is ready to run (if there is such a process) avoids wasting valuable CPU time unnecessarily. On the other hand, giving the CPU to a process consisting of a single, long, CPU burst until it runs to completion will maximise CPU utilisation, but it will also delay the execution of any other processes in the system, which may be undesirable.

The following assumptions and recommendations should be taken into account:

- Consider different process scheduling algorithms that, given a list of processes characterised by their sequences of CPU and I/O burst sizes (durations), decide which process should run on the CPU at each time until all processes run to completion. The implementations of four CPU scheduling algorithms are provided together with this document: First Come First Served (FCFS), Round Robin (RR), Shortest Job First (SJF) and Shortest Remaining Time First (SRTF) are provided together with this document. Note that the last two algorithms require the duration of the next CPU burst to be known in advance, or at least predicted, which may be unrealistic in computer process scheduling. The I/O scheduler always follows a first come first served mechanism regardless of the CPU scheduling strategy.
- The four schedulers are implemented in Python using the Salabim (www.salabim.org) discrete-event simulation package. The simulator reads a text file from `stdin` as input containing one line per process to be scheduled each containing a sequence of floating-point numbers of even length. In each line, the first number represents the arrival time of the process, and the remaining numbers represent the length of the CPU and I/O bursts that result from running the process. Since the processes must start and end with a CPU burst, the total number of bursts must be odd (and the number of values in each line must be even).

Each particular CPU scheduler is selected by passing appropriate command-line argument(s) to the simulator.

- There are also two files named `gen_workload.py` and `gen_workload.R` that can generate input files in the format described above. You can specify several command-line arguments in order to generate different sets of processes (workloads) with different characteristics, but ultimately you may also modify that file in order to be able to generate completely different workloads.
- The simulator outputs to `stdout` one line per process, each containing a sequence of numeric values separated by spaces. The first value denotes the id of the process, which is defined by the order of the processes in the input. The second value denotes the arrival time of the process to the scheduling system. The following three values give the sum of all CPU bursts, all I/O bursts, and all bursts, respectively. The final three values give the Turn Around Time (TAT), i.e. the total time the process spent in the system from its arrival to its completion, the Ready Wait Time, i.e. the time the process spent in the CPU scheduling queue ready to run, and the I/O Wait Time, i.e. the time the process spent waiting for I/O requests to be fulfilled.
- The experiments can be repeated using different conditions in order to understand the situation better, and allow the generalization of the results. It is up to you to explore this possibility.

2 Outcome

There are two milestones, each covering a particular technique discussed during the course:

1. Exploratory data analysis
2. Inferential data analysis

The goal of each milestone is to design the experiment, collect the data, and use the corresponding data analysis technique to draw conclusions about the problem statement. The main outcome of each milestone is a written report (PDF file, maximum 8 single-column pages, including cover, references and appendices). The report should describe the steps of the design of the experiment with enough detail to allow others to reproduce the experiment and, of course, should provide clear and sound conclusions. Given that the main outcome of the assignment is a report, the quality of the document (i.e., structure of the report, precision of the results reported, quality of writing) is of paramount importance. There is no suggested template for the report structure. The goal is to avoid the rather passive situation in which the students just fill in a given report structure. On the contrary, defining the best structure for the report is part of the goals of the assignment. The teachers are fully available to discuss the proposed structure with the students, as well as any other aspect of the report. Some of the Practice Lab sessions are specifically devoted to providing such support to students. In addition to the report, the students must submit a folder with all the programs and tools used in the work.

3 Resources

Students are expected to use their own computers.

4 Calendar and miscellaneous

The assignments must be carried out in groups of up to 3 students according to the following calendar:

- October 2, 2020 – Submit the names and student numbers of the elements of your group through inforestudante. Note that the groups must remain the same for all the assignments.
- October 30, 2020 – Submit the assignment report for the first milestone.
- December 11, 2020 – Submit the assignment report for the second milestone.
- Oral defence in January (to be defined later on for each group).

Submissions of the reports should be uploaded at inforestudante. Submissions after the deadline (at 23h59) shall not be allowed.

Plagiarism means mandatory fail in the course and possibly an internal (UC) disciplinary procedure. Please, refer adequately to all text and material you take from the Internet or other sources. All parts of the report must be written by the students and not copied, pasted and changed from the Internet.