

ECE 459: Programming for Performance

Assignment 4

Jeff Zarnett & David Cheng

March 17, 2018 (Due: April 4, 2018)

In this assignment, you will be asked to simulate a multi-server application to investigate the impact of Load Balancing on the system's overall performance. The goal here is both to implement load balancing, and also to make some determinations about what impact load balancing has (big or small, positive or negative) based on the parameters of the system.

A basic overview of the program as provided is: some number of jobs will arrive periodically (according to a Poisson distribution) to the system and they will be assigned to one of N queues either by (a) random assignment or (b) round-robin (i.e., job i is assigned to queue i modulo N). Each queue will have an associated `pthread` that will dequeue the first job, if any is available, and dispatch it to a processing function. When processing is complete, it outputs some timing data to the results file in a specified format, giving us some information about how long it takes to complete the processing of the task.

Getting started. Get your starter code from `git.uwaterloo.ca`.

To submit, push the C file containing your code, along with your report. The report will explain your load balancing strategy, and overview when load balancing is effective and when it is not.

This program takes a lot of command line arguments so you have lots of space to experiment with the program and evaluate it. The parameters are:

Character	Meaning
n	Number of "servers" (queues)
a	Assignment policy (1 for random, 2 for round robin)
j	Number of jobs
b	Load balancing (0 for off, 1 for on)
l	Lambda (arrival rate parameter)
m	Max Rounds (controls variability of job size)

The output file will be comma separated values ("results.csv"), with one line per completed job. The column order:

1. Job ID
2. Arrival time (the time at which the job enters the queue)
3. Execution time (the amount of time the job took in the `execute` function)
4. Departure time (when the job's execution is finished)
5. Response time (departure time minus arrival time)

Load Balancing

The implementation side of load balancing is writing the functionality into the program. A high level explanation is that you need to migrate jobs from over-loaded queues to under-loaded ones. This keeps things “balanced” within some margin. You should (probably) run the load balancing technique periodically. How often you wish to run it is a design decision: if you run it too often, you’ll waste a lot of time shuffling jobs around; not often enough and you do not see enough benefit.

Each queue uses the First-Come First-Served (FCFS) policy (i.e., all work is considered to be of equal priority).

One strategy for implementation follows, though you are welcome (encouraged) to come up with your own. At each period, measure the average length of the queues; where the length of each queue is measured by the total number of jobs currently waiting in it. If the length of a particular queue is higher than the average, then it is considered as an over-loaded queue and as such you need to migrate some jobs from the end of that queue to the under-loaded queues of that tier. This process continues till you have balanced queues (all queues are approximately equal in length, depending on how many jobs there are).

List of things I don’t want you to do: (1) have a single queue for incoming jobs (as this trivializes the load balancing), (2) have > 1 thread per queue (as this can mean that threads sit with the work items and get blocked, meaning load balancing does nothing).

A naive implementation may have an infinite loop, frequently moving the same job between the queues over and over. This is something to avoid...

This last column of the output file is important; the real goal is to minimize the **average 90th percentile response time** of all jobs (through load balancing). You will want to modify the provided `loadbalance.c` to implement the load balancing and do it efficiently. Then it is time for the report.

To write the report, there are four cases that form the baseline scenarios, shown below:

1. Random assignment, no load balancing
2. Round robin assignment, no load balancing
3. Random assignment, with load balancing
4. Round robin assignment, with load balancing

This covers the variables a and b in the parameters. Find the average 90th percentile response time of all jobs in each case, and investigate how they are affected by changes in the arrival rate (λ) and variability of job size (m). Choose a reasonable value of j for your testing.

The content of the report will then say some things like “load balancing has an increasing positive impact correlated with an increase in $x...$ ”. Consider also the combinations of variables λ and m (i.e, are there any synergistic effects).

Your report should support your findings with some hard (measured) numbers and the use of some graphs to illustrate is recommended. Intuition or logic might say that a change in x results in the outcome of y , but it must be quantified. A graph that charts the effectiveness of load balancing (compared to baseline) with some increasing or decreasing variable is very convincing to the reader, for example.

Tastefully-presented data should make up about half of the report and your discussion and explanations the rest. Avoid chartjunk¹, and don’t imprison your numbers².

¹<https://en.wikipedia.org/wiki/Chartjunk>

²<http://betterposters.blogspot.ca/2012/08/the-data-prison.html>

Evaluation

The general principle is that correct solutions earn full marks. However, it is your responsibility to demonstrate to the TA that your solution is correct. Well-designed, clean solutions are therefore more likely to be recognized as correct. Solutions that do not compile will earn at most 39% of the available marks. Segfaulting or otherwise crashing solutions earn at most 49%.

Your program will be evaluated over several runs to prevent bad luck on a single run from being the deciding factor. Your program should not leak memory, nor should it have data races that result in invalid or incorrect output.

Implementation (15 marks) A correct implementation is important. You may use any techniques to implement the load balancing, as long as it produces the right output, does not trivialize the work, and balances the load.

Report (35 marks) 30 marks for explaining the impact of load balancing (4ish pages is appropriate length); 5 marks for clarity of exposition.