IN5570v23 – Distributed Objects
Mandatory Assignment 2 (Oblig 2)
Breakeven and Distributed Time

Eric Jul
Department of Informatics
University of Oslo

February 23rd, 2023

Introduction

Exercise 1 – Break-even for call-by-visit

Call-by-visit may provide a performance improvement. However, if the parameter to an invocation is never called, call-by-visit is a waste of time and negatively affects performance. On the other hand, if the parameter is accessed billions of times during the call then call-by-visit has a positive influence on performance even when the parameter object is large. So for a given parameter object, there is a point, a specific number of calls, between zero calls and one billion calls where call-by-visit performance changes from a negative impact to a positive impact. Such a point is called a break-even point. For small objects, the break-even point is usually around one (when call-by-visit works as it should; see below for problems with this).

> Unfortunately, call-by-visit and call-by-move do NOT work properly in our current implementation (such is life with a non-commercial, research language). You are to emulate call-by-visit using explicit moves of the parameter object. Such emulation will be described in the class – and sample code will be put on the web site. Obviously, such emulation has an effect on the break-even point – however, it does not change the overall experiment nor its conclusion, so just report the results using this work-around.

Write and run a program to find the break-even point for the emulated call-by-visit with objects of a given size. Find break-even for a parameter object that has an attached instance variable referring to an array of integers with the following number of elements:

- 50
- 100
- 500
- 1,000
- 10,000

Experiment and find the break-even points. Explain why the array must be attached.

(Beware: the subject of *break-even point* will likely be one of the exam questions at the oral exam at the end of the course.)

Exercise 2 – Time Collector

Write and run a program like `kilroy` that visits each active node and collects the local time in an array. Test the program locally on at least four machines. Run on at least four different machines in at least four different countries on Planetlab preferably at least 100 km apart. Upon return, the program should print the collected times along with the node information (*e.g.*, node number and name, whatever...)

Exercise 3 – Time Synchronization

Write a program that puts a time agent on a number of nodes and then at regular intervals (for testing, *e.g.*, once a minute) makes a call to all its time agents to collect the time on each machine. Using a time synchronization method of your choice, compute a time that represents a reasonable approximation of the actual time. For each computed actual time, print the time and the difference for each of the times on the other nodes. Your program should work despite some of the nodes running the agents are inaccessible or down, *i.e.*, you should use `unavailable` handlers.

Test the program locally on at least four machines.

Test your program on Planetlab with at least four machines in at least four countries. The more nodes, the merrier... (but four is sufficient)

Explain the algorithm that you use for time synchronization. You are free to pick whatever algorithm that you fancy as long as you provide good arguments for your choice. Simple is good. advanced is for the ambitious—and those taking IN9570.

Additional requirements for IN9570 students

IN9570 students should compare their chosen algorithm to algorithms found in the literature and should cite and describe the essence of at least three articles that discuss time synchronization, *e.g.*, articles about how ntp works.

Delivery in Devilry

The assignment must be submitted in devilry.

In addition to the source code, you should deliver a PDF containing a text file with the output from a run of the programs and a short report (2-5 pages for IN5570 students; 5-8 pages for IN9570 students) commenting upon your decisions and results. When relevant, feel free to include graphs showing your results. When relevant, explain your choice of how many iterations of the test that you ran and what kind of result, you are reporting, *e.g.*, min/max, average values, median values, *etc.*

DEADLINE

The deadline is set to **March 15th, 2023 23:59:00 CET**.

Have fun!