CSC_4SL05_TP: Project
Optimistic Lock-Based List-Based Set Implementations

The goal of this project is to get familiar with optimistic fine-grained locking schemes in concurrent data structures. Our running example is *sorted linked list* used to implement a *set* abstraction.

The project is to be taken in teams of up to two students.

Synchrobench

To study the performance of three lock-based set implementations, we are going to use the `synchrobench` benchmark (`https://github.com/gramoli/synchrobench`).

In the simplest setting, `synchrobench` allows you to define a workload varying the following parameters:

- number of threads,
- size of the list,
- range of the list values,
- duration of the experiment,
- and fraction of updates (*add* and *remove*, equally shared).

In the simulated runs, every update picks a random value in the range uniformly at random.

The collected statistics contains the throughput (the number of complete operations per second), as well as the fractions of successful *remove*, *add* and *contains* operations.

Check INSTALL for instructions on adding new algorithms to the `synchrobench` library (you may also use `eclipse` with `build.xml` to do this), and README.md for the parameters of the simulations.

Algorithms

The goal is to analyze performance of several java implementations of a list-based set. We focus on the following algorithms:

- Coarse-grained locking (already present as `linkedlists.lockbased.CoarseGrainedListBasedSet.java`)
- Hand-Over-Hand locking
- Lazy Linked List by Heller et al. (already present as `linkedlists.lockbased.LazyLinkedListSortedSet.java`)

For the hand-over-hand algorithm refer to the slides or Chapter 9 of the book by Herlihy and Shavit (`https://www.dawsonera.com/abstract/9780123977953`).

Your first task is to implement the Hands-over-Hand algorithm in java. You may take the coarse-grained locking algorithm as a basis. Make sure that your new classes implement `AbstractCompositionalIntSet` and export the right interface.

Argue that your implementation is correct, i.e., linearizable and deadlock-free.

Experiments

A sample command to run `synchrobench` with a lock-based linked list algorithm ALG on 10 threads, update ratio 10, list size 1024, range 0-2047, and duration 3000ms is:

```
> java -cp bin contention.benchmark.Test -b linkedlists.lockbased.ALG -d 3000 -t 10 -u
  10 -i 1024 -r 2048
```

Run the three algorithms, coarse-grained, hand-over-hand and lazy, through `synchrobench` with the following parameters:

- Number of threads: 1,4,6,8,10,12
- Update ratios: 0, 10, 100
- List sizes: 100, 1k, 10k (choose the ranges of the lists twice as big, to maintain the expected list size constant)
- For the other parameters, specify `-W 0 -d 2000` (no "warmup", duration 2s).

It is important to use a real multiprocessor (with at least 12 hardware cores). You may use a machine in `http://lames.enst.fr/`, just check its current utilisation. (**Running your experiments on your dual-core laptop does not make any sense.**)

The current version of synchrobench requires Java 8, you might have to install this version first.

Report

Prepare a short report (up to 15 pages), preferably in English (can also be written in French if English does not feel comfortable).

The report should contain:

- Java code for your implementations of the hand-over-hand algorithm;
- A proof sketch of correctness for the hand-over-hand algorithms (argue that both safety and liveness hold);
- Performance analysis.

For the performance analysis, prepare a graph depicting the throughput as a function of the number of threads for the three algorithms, for some representative list size and update ratio. You may use `gnuplot` or any other plotting program of your choice.

Then, for each algorithm, fix the update ratio to 10%, and prepare a graph depicting the throughput as the function of the number of threads, varying the list size.

Finally, for each algorithm, prepare a graph depicting the throughput as the function of the number of threads, varying the update ratio, for the list size 1k.

Altogether, this gives:

- Three plots (one per algorithm), with three curves each, for a fixed update ratio 10% and varying list size.
- Three plots (one per algorithm), with three curves each, for a fixed list size 100 and varying update ratios.
- One plot, with three curves (one per algorithm), with fixed update ratio 10% and list size 1000.

Explain the forms of the curves and their relations to each other.

Include in your report the system details of the machine you used for your experiments.

The report (in pdf) should be submitted via ecampus (or, in case of emergency, by email, petr.kuznetsov@telecom-paris.fr).