Programming Assignment for Concurrency & Multithreading 2016-2017

Overview

The programming assignment consists of implementing two data structures in Java using different concurrency mechanisms, empirically examining their performance, and writing a report about the implementations and experiments. The data structures that should be implemented are the singly linked list and the binary search tree. Examples of list data structures can be found in [3] or [5], information on binary search trees can be found in [2] or [4].

The assignment is split into three tasks. Bachelor students must complete the first and second task, whereas Master students must complete all three tasks:

1. Implementing the data structures using coarse-grained locking algorithms.
2. Implementing the data structures using fine-grained locking algorithms.
3. Implementing the data structures using lock-free algorithms.

Details of these tasks, and a time schedule, can be found below. Note that the time schedule also includes a schedule for the feedback, so you can incorporate the feedback in the next task and your study of the material for the exam.

Prerequisites

First, decide whether you want to do the assignments alone or with someone else (you are encouraged to work in pairs). Then, you need to register for the assignment. You can do this by sending an email to Ceriel: `c.j.h.jacobs@vu.nl`. Include your name, email address, and student-number, and those of your partner if applicable. Make sure to also indicate whether you are a Master or Bachelor student and whether you already have a DAS4 account. If you don't, we will provide you with a temporary one.

On the course website at `http://www.cs.vu.nl/~tcs/cm/` you can find the papers referenced by this document and a framework for your implementation.

You will be using a standard compute node of the DAS4 cluster [1] to test and evaluate your solution. Access to the clusters requires a DAS4 account, see above.

During development you can also use your own computer systems, but your implementation should always be able to run on a DAS4 node. Make sure your machine meets the following requirements:

- A Java JDK version 1.7 or higher is installed. You need a JDK (Java Development Kit), a JRE (Java Runtime Environment) is not sufficient.
- The program `ant` is available to build your implementation.

[1]`http://www.cs.vu.nl/das4/`

- Multiple cores are available for testing.

Also keep in mind that different operating systems can have different performance characteristics, so be sure to double-check your intermediate results on a DAS4 node. Final measurements for the evaluation part of the assignment should always be done on a DAS4 node.

Structure

Each task requires you to

- design and implement a synchronization scheme;
- create hypotheses about performance;
- and evaluate the performance of your data structures.

Implementation Implement your data structures in Java using the supplied framework. All data structures implement the `Sorted` interface which provides the methods `add(T t)`, `remove(T t)`, and `toArrayList()`. The data structures need to be sorted according to the ordering of `Comparable<T>`[2]. We provide a basic skeleton that uses the discussed data structures. It is your task to implement the `add(T t)`, `remove(T t)`, and `toArrayList()` methods according to the requirements (see below). Use a clear coding style.

Hypotheses Hypothesize about the performance and scalability of your data structure. Try to qualitatively estimate the performance of the data structures by taking into account the number of elements in the data structure, the number of threads operating on the data structures, and the amount of work a thread is doing in comparison to inserting or removing an element. Relate to previous data structures if possible. Make sure to support your claims.

Evaluation Evaluate the performance of your data structures using a node of the DAS4 cluster. Intermediate testing of your implementation should mostly be done on your local machine, but you can of course do a couple of test runs on a DAS4 node, because such a node may have more cores and that may affect the behaviour of your data structures. In the evaluation section of your report you assess the performance of the data structures and compare them to both your earlier stated hypotheses, and the performance of previous data structures. Your evaluation should be supported with quantitative measurements, for example using graphs and tables. Use reasonable element counts (a sequential run (a run with 1 thread) should take somewhere between a couple of seconds and a couple of minutes to allow you to measure speedups. Also use reasonable numbers of threads, considering the host that you run your measurements on. Explain your results. Do the measurements correspond with your expectations? And if not, why not?

Task 1: Coarse-grained data structures

For this task you are required to implement the two data structures using coarse-grained locking. Coarse-grained locking means that you first write a sequential implementation, and then ensure that only one thread can access your data structure simultaneously.

- Implement the coarse-grained data structures using the framework.
- Hypothesize on the performance of the coarse-grained data structures.
- Evaluate the performance of the coarse-grained data structures.

[2]`http://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html`

CoarseGrainedList This CoarseGrainedList needs to be a singly-linked list. The elements are ordered in a monotonically non-descending order. This means that the list may contain duplicate elements. Adding or removing an element should lock the complete data structure. Figures 9.4 and 9.5 in [3] show examples of update methods of a linked list with coarse-grained locks.

CoarseGrainedTree This tree is a binary search tree that allows duplicate elements. Elements are stored at internal nodes and leaf nodes of the tree. The binary search tree does not have to re-balance itself. The complete data structure needs to be locked when an element is added or removed.

Task 2: Fine-grained data structures

For this task you are required to implement the two data structures using fine-grained locking, See section 9.5 of [3]. Fine-grained locking is much more error-prone than coarse-grained locking, so test your implementation thoroughly! We will do so as well ...

- Implement the fine-grained data structures using the framework.
- Hypothesize on the performance of the fine-grained data structures.
- Evaluate the performance of the fine-grained data structures.

FineGrainedList This singly-linked list should not lock the complete data structure, but only parts of it to allow concurrent updates on different parts of the list. Figure 9.6 and 9.7 in [3] show examples of updates on a list with fine-grained locks.

FineGrainedTree This binary search tree is similar to a CoarseGrainedTree, but uses fine-grained locks, so locks at the nodes of the tree. Concurrent updates on different parts of the tree should be allowed.

Task 3: Lock-free data structures

Note: This task can be very challenging, and is only required of Master students!

For this task you are required to implement the two data structures using lock-free synchronization. Lock-free synchronization can be accomplished in Java using the java.util.concurrent.atomic package[3].

- Implement the lock-free data structures using the framework.
- Hypothesize on the performance of the lock-free data structures.
- Evaluate the performance of the lock-free data structures.

LockFreeList This lock-free linked list should (obviously) not use locks at all. Section 9.8 in [3] discusses an example of a lock-free list.

LockFreeTree Implementing a lock-free binary search tree is a challenging task so to make it somewhat easier (?), duplicate elements should no longer be supported. Ellen et al. discuss a leaf-based (values are only stored in leaf nodes) implementation in [1]. You should implement this lock-free binary search tree, following the algorithm as described in the paper.

[3]`http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/atomic/package-summary.html`

Framework

There are two main differences between the data structures in [3] and the data structures required by the assignment. First, the list data structure presented in the book is sorted according to the hash value of the stored items, whereas we ask you to sort the list according to the `Comparable<T>` interface. Furthermore, the data structures in the book do not allow duplicate elements, whereas all data structures in this assignment should allow duplicate elements, except for the lock-free tree.

The `toArrayList()` method should return an ArrayList containing the elements of the data structure, in the order indicated by the data structure. It does not have to be thread-safe. You should focus on implementing the `add()` and `remove()` methods. You may assume that `toArrayList()` will not be invoked during concurrent updates.

We have provided a run script located in `bin/test_data_structures` to start the framework. The framework will first create a specified number of threads, have these threads add elements in random order, synchronize the threads using a barrier, and have the threads remove the same elements in a different order. After this, the data structure should be empty. Executing the script without parameters displays a short manual.

In order to ease the debugging process, the framework seeds the random number generator based on the arguments you pass to it, ensuring that the same set of parameters always results in the same input to your algorithm.

The script detects when it is run on the head node of the DAS4 cluster, and will automatically run the framework on one of the DAS4 nodes instead.

Timeline

The tasks must be submitted before 23:59 on the day they are due. You will receive feedback within a week.

Deadlines are strict! If you are unable to finish the assignment in time, at least submit your unfinished work before the deadline.

Deadlines for bachelor students

#	Task	Date	Feedback due
1	Coarse-grained	November 20th	November 27th
2	Fine-grained	December 11th	December 18th

Deadlines for master students

#	Task	Date	Feedback due
1	Coarse-grained	November 13th	November 20th
2	Fine-grained	November 27th	December 4th
3	Lock-free	December 11th	December 18th

Assessment criteria

Your final grade depends on the quality of your implementation, your hypotheses and their justification, and the thoroughness of your evaluation, but also on the completeness of your intermediate submissions. Please make sure that your code can be easily understood and uses a clear coding style. Your report for each task should be approximately 2-3 pages of text (not including raw measurement data and graphs).

Submission procedure

The report document should be in PDF format. Your java source files need to be compressed in a single archive, for which you can use a helper program named `submit.sh` in the bin directory of the framework. Note that your program should run without any modification to the original framework. In particular, you may not change the `Sorted` interface, since we will test your implementations in ways not provided in the framework. You are allowed (and encouraged) to modify the framework for e.g. debugging purposes, but we will use the original framework during the evaluation of your code.

Always list both the VU-net-ID and student-number of you and your partner (if applicable) in your submissions. Your submission mail should contain two attachments: the report in PDF format and the archive as delivered by the `submit.sh` script. Send your submission to `c.j.h.jacobs@vu.nl`.

References

[1] Faith Ellen, Panagiota Fatourou, Eric Ruppert, and Franck van Breugel. Non-blocking Binary Search Trees. In *Proceeding of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '10, pages 131–140, New York, NY, USA, 2010. ACM.

[2] Michael T Goodrich and Roberto Tamassia. *Algorithm Design: Foundation, Analysis and Internet Examples.* John Wiley & Sons, 2006.

[3] M. Herlihy and N. Shavit. *The Art of Multiprocessor Programming.* Morgan Kaufmann Pub, 2008.

[4] Wikipedia. Binary Search Tree (`http://en.wikipedia.org/wiki/Binary_search_tree`). [Online; accessed 20-October-2016].

[5] Wikipedia. Linked List (`http://en.wikipedia.org/wiki/Linked_list`). [Online; accessed 20-October-2016].