Lab 12: Java Locks

Instructor: Vivek Sarkar, Due: Friday April 15, 2015

Resource Summary

Course wiki: `https://wiki.rice.edu/confluence/display/PARPROG/COMP322`

Staff Email: comp322-staff@mailman.rice.edu

Important tips and links:

edX site : `https://edge.edx.org/courses/RiceX/COMP322/1T2014R`

Piazza site : `https://piazza.com/rice/spring2015/comp322/home`

Java 8 Download : `https://jdk8.java.net/download.html`

Maven Download : `http://maven.apache.org/download.cgi`

IntelliJ IDEA : `http://www.jetbrains.com/idea/download/`

1 Lab Goal

In today's lab you will practice using Java Locks.

The Maven project for this lab is located in the following svn repository:

- `https://svn.rice.edu/r/comp322/turnin/S15/`*`NETID`*`/lab_12_locks`

Use the subversion command-line client to checkout the project into appropriate directories locally. For example, you can use the following commands from a shell:

```
$ cd ~/comp322
$ svn checkout https://svn.rice.edu/r/comp322/turnin/S15/NETID/lab_12_locks/ lab_12
```

In today's lab, you need to use STIC to submit computation jobs. If you need any guidance on how to submit jobs on STIC, please refer to the appendix at the end of this document or any of the previous labs in which you used STIC.

You also need to configure arguments for your program. If you forget how to do so, please refer to the note on page 1 of lab_9.

2 Sorted Linked List Example using Java's Synchronized Methods

In the repository you'll find 8 java files: `SyncList.java`, `CoarseList.java`,`RWCoarseList.java`, `ListDriver.java`, `ListCounter.java`, `ListSet.java`, `ListTest.java`, `RWMix.java`. Of these, you only need to focus on `SyncList.java`, `CoarseList.java`, and `RWCoarseList.java`. `SyncList.java` contains a thread-safe implementation of a sorted linked list that supports `contains()`, `add()` and `remove()` methods. The default driver options repeatedly calls these three methods with a distribution that aims for 98% read operations (calls to `contains()`), 1% add operations, and 1% remove operations. Since all three methods are declared as `synchronized` in SyncList.java, all calls will be serialized on a single `SyncList` object.

For this section, your tasks are as follows:

1. Compile and run the programs locally as following to test your program
   ```
   mvn clean compile exec:exec -Plistdriver
   ```
2. Then on STIC, modify the myjob.slurm template that has been provided to you under:
   `lab_12_threads/src/main/resources`
   to submit lab 12 computation jobs to the computing node.
   Observe the performance reported next to the text "Operations per seconds:". Since this is a throughput metric, a larger value will indicate better performance. How does the performance vary with number of threads? Can you explain why this happens?

3 Use of Coarse-Grained Locking instead of Java's Synchronized Methods

The goal of this section is to replace the use of Java's synchronized method in `SyncList.java` by explicit locking instead. For this section, your tasks are as follows:

1. Copy contents of the constructor and three functions from `SyncList.java` into `CoarseList.java`.
2. Allocate a single instance of `ReentrantLock` when creating an instance of `CoarseList`.
3. Replace the three occurrences of "synchronized" by appropriate calls to `lock()` and `unlock()`. Remember to use a try-finally block as follows to ensure that `unlock()` is always called:
   ```
   lock.lock();
   try { ... }
   finally { lock.unlock(); }
   ```
4. Compile and run the programs locally as following to test your program
   ```
   mvn clean compile exec:exec -Plistdriver
   ```
5. Then on STIC, submit lab 12 computation jobs to the computing node.
   How does the performance compare with the performance observed for `SyncList`?

4 Use of Read-Write Locks

The goal of this section is to replace the use of a `ReentrantLock` in `CoarseList.java` by a `ReentrantReadWriteLock`, so as to leverage the fact that the majority of the operations (98% by default) are calls to `contains()` which are read-only in nature. For this section, your tasks are as follows:

1. Copy contents of `CoarseList.java` into `RWCoarseList.java`.
2. Replace the instance of `ReentrantLock` by an instance of `ReentrantReadWriteLock`.
3. Replace the calls to lock() by readLock.lock() or writeLock.lock() where appropriate. Likewise for unlock().
4. Compile and run the programs locally as following to test your program
   ```
   mvn clean compile exec:exec -Plistdriver
   ```
5. Then on STIC, submit lab 12 computation jobs to the computing node.
   How does the performance compare with the performance observed for `CoarseList`?

5 Turning in your lab work

For each lab, you will need to turn in your work before leaving, as follows.

1. Show your work to an instructor or TA to get credit for this lab. Be prepared to explain the lab at a high level.
2. Check that all the work for today's lab is in the `lab_12_locks` directory. If not, make a copy of any missing files/folders there. It's fine if you include more rather than fewer files — don't worry about cleaning up intermediate/temporary files.
3. Use the svn command script to submit the `lab_12_locks` directory to your turnin directory as explained in the first handout. Note that you should *not* turn in a zip file.

Appendix: STIC setup

STIC(Shared Tightly-Integrated Cluster) is designed to run large multi-node jobs over a fast interconnect. The main difference between STIC and CLEAR is that STIC allows you to gain access to compute nodes to obtain reliable performance timings for your programming assignments. On CLEAR, you have no control over who else may be using a compute node at the same time as you.

- Login to STIC.

  ```
  ssh ⟨your-netid⟩@stic.rice.edu
  ⟨your-password⟩
  ```

  Your password should be the same as the one you have used to login CLEAR. Note that this login connects you to a *login* node.

- After you have logged in STIC, run the following command to setup the JDK8 and Maven path.

  ```
  source /home/smi1/dev/hjLibSource.txt
  ```

  Note: You will have to run this command each time you log on STIC. You could choose to add the command in /.bash_profile so that it will run automatically each time you log in.

- Check your installation by running the following commands:

  ```
  which java
  ```

  You should see the following: `/home/smi1/dev/jdk1.8.0_31/bin/java`

  Check java installation:

  ```
  java -version
  ```

  You should see the following:

  ```
  java version "1.8.0_31"
  Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
  Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07, mixed mode)
  ```

  Check maven installation:

  ```
  mvn --version
  ```

  You should see the following:

  ```
  Apache Maven 3.1.1 (0728685237757ffbf44136acec0402957f723d9a; 2013-09-17 10:22:22-0500)
  Maven home:  /home/smi1/dev/apache-maven-3.1.1
  Java version:  1.8.0_31, vendor:  Oracle Corporation
  Java home:  /home/smi1/dev/jdk1.8.0_31/jre
  Default locale:  en_US, platform encoding:  UTF-8
  OS name:  "linux", version:  "2.6.18-371.perfctr.el5", arch:  "amd64", family:
  "unix"
  ```

- When you log on to STIC, you will be connected to a *login node* along with many other users. Once you have an executable program, and are ready to run it on the compute nodes, you must create a job script that uses commands to prepare for execution of your program. We have provided a script template in

  ```
  lab_12_threads/src/main/resources/myjob.slurm
  ```

- To submit the job, run the following command in the same directory where you put myjob.slurm(in this case, it was place under `lab5/src/main/resources/myjob.slurm`):

  ```
  sbatch myjob.slurm
  ```

  After you have submitted the job, you should see the following:

  ```
  Submitted batch job [job number]
  ```

- To check the status of a submitted job, use the following command:

  ```
  squeue -u [your-net-id]
  ```

- To cancel a submitted job, use the following command:

  ```
  scancel [job-id]
  ```

  When your job finished running, your should see an output file titled `slurm-[job-id].out` in the same directory where you have submitted the job.

- To transfer a project folder to STIC, you can use one of two methods:
  - Use Subversion: You can commit your local changes to SVN. Then you can checkout or update the project on your STIC account using one of the following:

    ```
    svn checkout https://svn.rice.edu/r/comp322/turnin/S15/NETID/lab_12/
    ```

    or, if you have already checked out the SVN project on your account,

    ```
    svn update
    ```

  - Use SCP: Use the following command on your local machine:

    ```
    scp -r [folder-name] [your-net-id]@stic.rice.edu:[path to the storage location]
    ```

    For example, if I have a folder named ”lab12” on my local machine, and I want to store it in “./comp322” on STIC, I would type the following command:

    ```
    scp -r lab11 [net-id]@stic.rice.edu:./comp322
    ```