**Nicholas Thomas**

**CLC-Mini\_Project\_6**

**Question-2**

**Dr. Toure**

**September 16, 2018**

**Question 2**

Describe 3 factors that can slow down execution of Java applications, and which can be alleviated by the use of parallel programming. Substantiate your answer with code examples and screenshots of execution. Seek instructor guidance as needed.

Factors that can cause a slow down in the performance of a java application are:

1. The amount of threads you have in the thread pool of the application. This can cause the application to slow down drastically depending on the number of tasks being asked to execute. If you are trying to accomplish more task than the allowed number of thread pools, the remaining tasks will be put in a waiting state till an available thread is open to process the next task.
2. Logging is another reason your application can slow down. This can cause more problems than just slowing down your application. It can lead to filling up the disk drives of the computers, which over time can cause the application logging to stop when space becomes unavailable. And it is recommended to minimum logging such as ‘INFO’ or ‘WARN’ log4j levels. To reduce the amount of logging being down.
3. Application design can also be a factor in slowing down the performance of the application. This is something that may not be able to be completely controlled due to the nature of the build and is almost inevitable. Some causes can include unnecessary data being stored in the in the HTTP ‘Session’ or when using JMS the unnecessary use of ‘persistent’ messages.

Which can be Alleviated with Parallel Programming:

From what I have read and the TDQ that were covered in this week’s curriculum, I think that my first factor of not having enough threads would be the most logical one to use parallel programming on. For it would allow the application to be broken down into a bunch of smaller tasks, which theoretically in return cause the application to process the task a little quicker and not leave that many tasks sitting in a waiting state.

Parallel Programming Example:

The Example I have chosen was one found from the internet, the Reasoning for this is that I was never able to successfully install Glassfish server into my Eclipse IDE. So I made the decision to switch to Netbeans and while there are both programming environments I have never used it so am trying to navigate it still.

import java.util.ArrayList;

import java.util.List;

import java.util.Random;

import java.util.concurrent.Callable;

import java.util.concurrent.ExecutionException;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Future;

public class ExecutorServiceExample {

private static final Random PRNG = new Random();

private static class Result {

private final int wait;

public Result(int code) {

this.wait = code;

}

}

public static Result compute(Object obj) throws InterruptedException {

int wait = PRNG.nextInt(3000);

Thread.sleep(wait);

return new Result(wait);

}

public static void main(String[] args) throws InterruptedException,

ExecutionException {

List<Object> objects = new ArrayList<Object>();

for (int i = 0; i < 100; i++) {

objects.add(new Object());

}

List<Callable<Result>> tasks = new ArrayList<Callable<Result>>();

for (final Object object : objects) {

Callable<Result> c = new Callable<Result>() {

@Override

public Result call() throws Exception {

return compute(object);

}

};

tasks.add(c);

}

ExecutorService exec = Executors.newCachedThreadPool();

// some other exectuors you could try to see the different behaviours

// ExecutorService exec = Executors.newFixedThreadPool(3);

// ExecutorService exec = Executors.newSingleThreadExecutor();

try {

long start = System.currentTimeMillis();

List<Future<Result>> results = exec.invokeAll(tasks);

int sum = 0;

for (Future<Result> fr : results) {

sum += fr.get().wait;

System.out.println(String.format("Task waited %d ms",

fr.get().wait));

}

long elapsed = System.currentTimeMillis() - start;

System.out.println(String.format("Elapsed time: %d ms", elapsed));

System.out.println(String.format("... but compute tasks waited for total of %d ms; speed-up of %.2fx", sum, sum / (elapsed \* 1d)));

} finally {

exec.shutdown();

}

}

}

Resources:

Subramanian, K. (2015, December 16). Top 10 reasons why your Enterprise Java Application is slow. Retrieved from <http://karunsubramanian.com/websphere/top-10-reasons-why-your-enterprise-java-application-is-slow/>

Code Example:

What is the easiest way to parallelize a task in java? (n.d.). Retrieved from <https://stackoverflow.com/questions/2016083/what-is-the-easiest-way-to-parallelize-a-task-in-java>