CS5010 Fall 2018

Assignment 6 - Basic Concurrency

Overview

The aim of this assignment is to give you experience designing and implementing a concurrent program to process a reasonably large amount of data. We'll start by implementing a serial version of the problem, and then redesign to create a highly concurrent solution.

The Problem Scenario

Your overworked and underpaid Professor is desperately trying to write a research paper on performance testing of cloud-based applications. He has generated a tonne of data from tests that is currently processed partially automatically, and partially manually in a spreadsheet. To speed up results processing, a fully automated solution is needed.

This is where you come in.

The current tests produce two very simple .csv files.

One records the results of HTTP POST requests sent to a server, and the other records the results of HTTP GET operations.

In both files, the format is identical, ie:
{time stamp, operation type, latency, HTTP response code}.

- Time stamp is a value returned from System.currentTimeMillis();.It represents the time the request was sent
- Operation is POST or GET
- Latency is a value in milliseconds - the round trip response time for the request.
- HTTP response code should be self explanatory :) It should be 200.

Here's a sample extract from the POST file:

154134849941	POST	1814	200
154134850122	POST	246	200
154134850147	POST	112	200
154134850158	POST	160	200

154134850174	POST	107	200
154134850185	POST	174	200

Your task is to process the this raw data from both files and produce performance statistics that characterize the test.

The processing has several steps:
First, for each file (POST and GET), process the individual request latencies into 1 second 'buckets'. Take the first time stamp in each file as time 0, and create buckets for each second that the test runs for, relative to the start time.

Write two new .csv files which contain the output of your processing, with the format:

{1 Second Bucket start ime,
# of requests in that bucket,
mean latency of requests in that bucket)

Here's an example of this format

1541348499	3	1087
1541348500	11	233
1541348501	12	363
1541348502	11	132
1541348503	16	145
1541348504	16	380

After writing these lines to the files, you need to calculate:

1. Test length: How many second between start of POST/GET and end?
2. Mean latency (in milliseconds, no decimal places)
3. The 99th percentile latency (in milliseconds, no decimal places)
4. Total Throughput = total number of requests processed/test length in seconds

Write these values at the bottom of each csv file after 'bucketizing' the results.

Next ….:

Each test sends more POSTs than GETs. So while you should see POST requests continuously through the whole test time (every bucket), there will be intervals when there are POSTs but no GETs. Here's an example, where there are no GET requests in time interval 1538758101 onwards.

POST	**GET**	**TOTAL**

1538758093	575	176	1538758093	34	88		609
1538758094	557	185	1538758094	27	94		584
1538758095	553	181	1538758095	17	88		570
1538758096	570	181	1538758096	12	78		582
1538758097	542	186	1538758097	11	88		553
1538758098	569	181	1538758098	13	78		582
1538758099	568	181	1538758099	10	101		578
1538758100	562	181	1538758100	9	79		571
1538758101	570	180					570
1538758102	579	170					579
1538758103	589	178					589
1538758104	551	182					551
1538758105	575	177					575

You therefore need to read through the two result files and produce a third with the above format. This file 'time aligns' requests so that POSTs and GETs for the same time interval appear on the same row, and the total number of requests for that interval is in a new column on the right. Insert a blank column after the POST and GET columns.

Finally, calculate:

1. The overall throughput for the test (combined POST and GET) in requests per second
2. The peak throughput - ie the bucket with the higher total throughput

And write these values at the bottom of the file.

Part 1 - Sequential Solution

Easy - write a sequential version to solve the above problem.

The main class should be called PerformanceDataProcessor() and accept two file names as command line arguments - the first contains the POST results, the second the GET results. These files are .csv file, and have a name that ends with the word 'raw'.

When you write the two new processed files, replace 'raw' with 'results'.
When you write the time-aligned file, replace 'results' with 'combined'.

For testing you are provided with two sets of 'raw' files. All have long test names, but the ones with *JavaTest* in the title are relatively small in size. The other pair are quite a bit larger :). This size may influence your solution.

Step 2 - Concurrent Solution

Now, redesign your solution to use threads in a 'producer-consumer' design style. Same inputs. Same outputs. But the code now contains multiple threads.

For example, every time you read a file, you can think of this as a producer that simply knows how to read this given file type and write each line ‘somewhere’ for another thread, the consumer, to process. Consumers could read the contents and transform the file contents into the results required, and write these ‘somewhere’ for maybe another thread to write the output. This is called a ‘pipe-and-filter’ architecture.

Your solution must have producer and consumer threads running concurrently, and communicating via some suitable shared data structure. Producers and consumers should be loosely coupled and communicate only through some queue or channel.

Step 3: Identify the Peak phase

Modify your concurrent solution to take an additional integer value on the command line. Let’s call this ‘threshold’. The threshold is a value that the user wants to use to identify the PEAK test load period, when the client load is at its highest.

You should use it as follows:

1. Process the {testname}POST-results.csv file to find the first interval with throughput >= to the threshold value. This interval marks the start of the peak phase.
2. Now find the end of the peak phase by finding the last interval with throughput >= to the threshold value.
3. Create a new file called .{testname}POST-peak.csv that contains all the rows from the peak phase
4. Calculate for the peak phase data the following, and write the results at the bottom of the file:
    a. The duration of the peak phase in seconds
    b. The mean throughput
    c. The highest interval requests/sec
    d. The 5th percentile value for the peak phase throughput
    e. The mean response time
    f. The 99th percentile response time

And that’s it!!

Submission Requirements

As per previous assignments, use the maven archetype. Write great code or you will lose points. **Deadline Monday 12th November 6pm PST**