Philip Quan

cs6650/assignment-01

https://github.com/thephilipquan/cs6650-assignment-01

`part01`

package = https://github.com/thephilipquan/cs6650-assignment-01/tree/main/src/main/java/org/philipquan/part01

My client consists of 1 class Step01.Main with the following main logic...

1. Call `main.initialRun()` where we create 10 threads that call the POST and GET API 100 times each.
2. Start stopwatch.
3. Process each group. Waiting `delayInSeconds` before starting the next group.
4. Wait for all threads to finish via `CountDownLatch.await()`.
5. Stop stopwatch.
6. Report statistics.

Some other things that happen in between some steps are steps like parsing the command line arguments and checking if the `hostURL` exists. Other than that, I don't have a web of classes to discuss their relationship or anything. I packaged most of the code in an instance of `Main` because I abstracted my code into layers and a lot of layers needed the same input. So, instead of passing it through every method's parameters, the methods just call `this.foo`.

Here is my data for part01.

part01

Client	Group Count	WallTime	Throughput
Java	10	98.245s	2036/s
Go	10	102.657s	1948/s
Java	20	120.960s	3307/s
Go	20	130.669s	3061/s
Java	30	142.964s	4197/s
Go	30	180.362s	3327/s

You can view the screenshots of the terminal output at my repo at https://github.com/thephilipquan/cs6650-assignment-01/tree/main/screenshots.

`part02`

package = https://github.com/thephilipquan/cs6650-assignment-01/tree/main/src/main/java/org/philipquan/part02

My approach to collecting statistics for the request latency involved the following...

- Fill a synchronized collection with instances of `MethodStatistic`. The collection is passed through the constructor and utilized in `Main.callServerMethod(...)`.
- Aggregate and report in `Main.reportStatistics()`.

An addition of 38 lines to `Main` along with `MethodStatistic` to hold the necessary information. That's pretty much it.

Here is my data for part02.

part02

Client	Group Count	WallTime (with stats)	Throughput (with stats)	min	max	mean	median	p99
Java	10	94.125s	2125/s	4ms	1047ms	37ms	37ms	59ms
Go	10	96.351s	2076/s	6ms	564ms	37ms	37ms	60ms
Java	20	123.240s	3246/s	3ms	1102ms	42ms	40ms	75ms
Go	20	129.634s	3086/s	4ms	557ms	43ms	40ms	132ms
Java	30	146.285s	4102/s	2ms	1092ms	43ms	42ms	76ms
Go	30	172.836s	3471/s	4ms	1056ms	56ms	43ms	221ms

You can view the screenshots of the terminal output at my repo at https://github.com/thephilipquan/cs6650-assignment-01/tree/main/screenshots.

And here is my chart comparing the wall time and throughput for client calls consisting of 30 groups for both part01 and part02.