ACS Programming Assignment 2

Anders Holst, wlc376

2020-12-18

0 Implementation

I implement all the specifications given in the assignment text. I omit it from the report since it is not specified to include documentation in the report, and I don't want to go too much over the page count.

In any case, my implementation is attached to the hand-out, and the source code is self-documenting via appropriate code comments.

---

1 Experiments

1.1 Experiment setup

- *Describe in detail the setup you have created for your experiments.*

Sample data generation

When generating random sets of `ImmutableStockBooks`, I of course want to generate books with different properties (title, price, number of sales misses, and so on).

I write a function that generates a single `StockBook`. The function randomly generates each of the parameters to the `ImmutableStockBook` constructor except for the ISBN, which is passed as a parameter to the function. This is to more easily generate sets of books with all unique ISBNs.

```
1  public StockBook nextStockBook(int isbn) {
2    String title  = "title_"  + isbn; // suffix books with ISBN to guarantee unique names.
3    String author = "author_" + isbn;
4
5    float price   = rand.nextFloat() * 50 + 0.1f;  // price from 0 to 50.
6    int numCopies = rand.nextInt(61) + 20;         // 20 to 80 copies.
7
8
9    long numSaleMisses = rand.nextInt(11);         // 0 to 10 sales misses.
10   long numTimesRated = rand.nextInt(41);         // 0 to 40 times rated.
11   long averageRating = rand.nextInt(6) + 1;      // average rating between 1 and 5.
12   long totalRating = numTimesRated * averageRating;
13
14   // one third of books is an editor pick.
15   boolean isEditorPicked = rand.nextInt() % 3 == 0;
16
17   return new ImmutableStockBook(isbn, title, author,
18           price, numCopies, numSaleMisses,
19           numTimesRated, totalRating, isEditorPicked);
20 }
```

`nextSetOfStockBooks()` then calls this in a loop over a pre-generated set of unique ISBNs.

Note that the title is simply the word "`title_`" suffixed with the ISBN. ISBNs are always chosen in the inclusive integer range $\{1,...,10^9\}$ *(to avoid collisions during generation), so titles are between 7 and 16 characters long. Author fields are generated in the same fashion, which means author strings are between 8 and 17 characters long.*

Workload configuration

When testing with large numbers of worker thread (>100), I found that the success rate would sometimes drop as low as 95%. To counter this, I modified some of the workload configuration parameters in the `WorkloadConfiguration` class.

Below snippet shows those configuration parameters that I changed:

```
client.workloads.WorkloadConfiguration.numEditorPicksToGet: 20 -> 10
client.workloads.WorkloadConfiguration.numAddCopies:        20 -> 8
client.workloads.WorkloadConfiguration.numBooksToAdd:       10 -> 4
```

With these changes, I never experienced failing interactions. However, this comes at the consequence that interactions are now *smaller*, in the sense that each interaction manipulates a smaller number of books. This must be considered in viewing the benchmark results.

No other configuration parameters are changed, meaning tests are still run 500 times (following 100 warm up runs).

Hardware and hyperparameters

All experiments (local and otherwise) are run on my local machine, featuring an intel i7-7700hq 2.8GHz CPU. My machine also has 8GB of RAM, which might be useful when the number of worker threads become large[1].

To test a varying number of worker threads, I have changed various hyperparameters in the source code to allow more simultaneous worker threads and HTTP proxy clients. Specifically, I have made the following changes:

```
client.BookStoreClientConstants.CLIENT_MAX_THREADSPOOL_THREADS: 250 -> 512

server.BookStoreHTTPSERVER.MIN_THREADPOOL_SIZE: 10  -> 1
server.BookStoreHTTPSERVER.MAX_THREADPOOL_SIZE: 100 -> 512

utils.BookStoreConstants.BINARY_SERIALIZATION: false -> true
```

Notice that I have also chosen binary serialization. This decision was made in an effort to speed up non-local tests, which quickly became excruciatingly slow for larger worker thread counts (> 100).

1.2 Experiment execution

I want to measure throughput and latency of the `CertainWorkload`. I want to benchmark using various numbers of worker threads (hereby denoted by N), and for this I have chosen $N = 2^i$ for $i \in \{0, ..., 9\}$, ie. all powers of 2 from 1 to 512.

[1] To be fair, I am not at all familiar with how or whether JVMs utilize 64-bit memory address spaces.

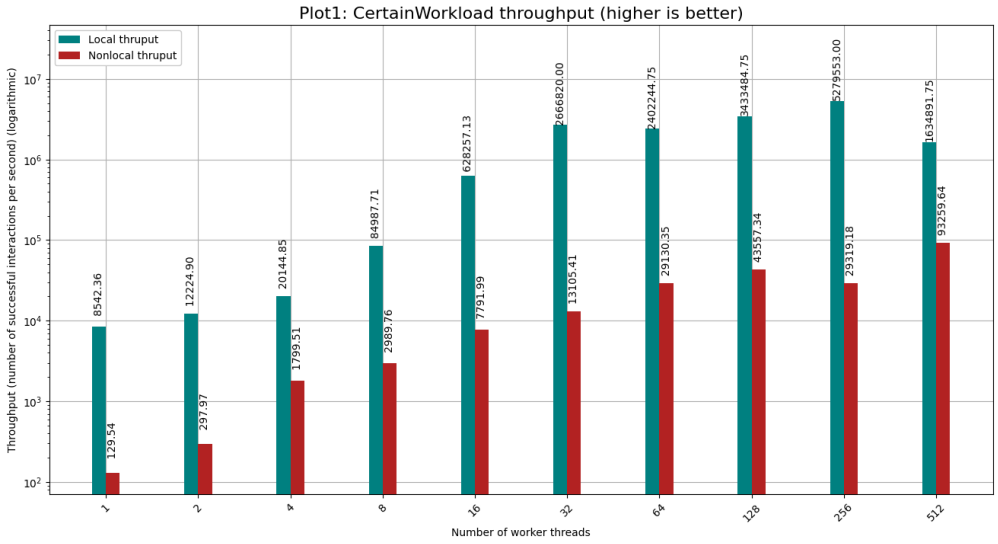
I execute all tests both locally and non-locally.

All tests are run on the hardware and under the circumstances described above.

1.3 Experiment results

1.3.1 Throughput measurements

Plot 1 shows the measured throughput:



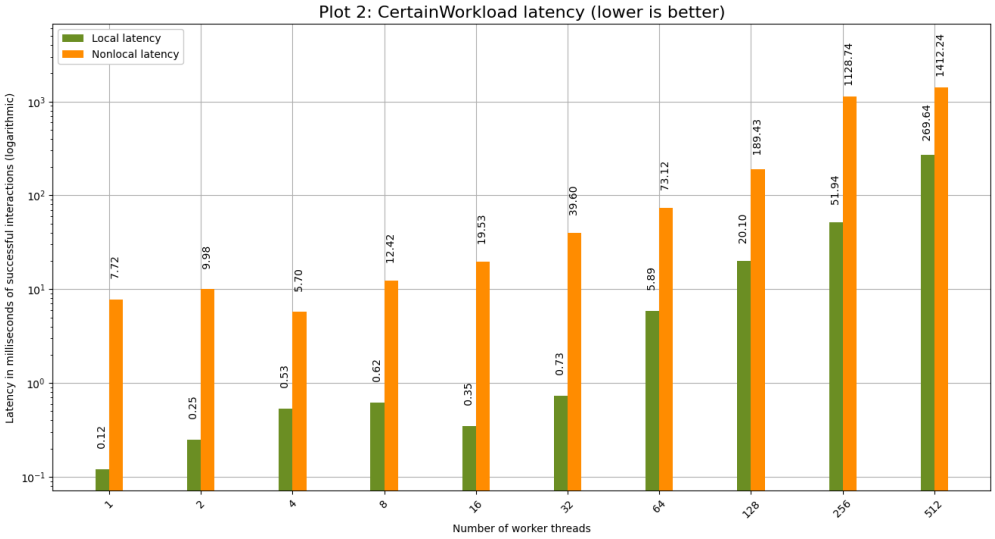
Please note that the plot is logarithmic.

The highest throughputs are measured for $N = 256$ and 512 for the local and non-local tests, respectively. As expected, throughput grows with N, but whereas there is a peak at $N = 256$ for local tests, the maximum throughput for non-local execution may require even larger N.

Not surprisingly, the local execution reaches throughput many orders of magnitude bigger than the non-local tests - this of course due to the immense overhead in RPC calls and the fact that here, the book store is run on an entirely different JVM.

1.3.2 Latency measurements

Plot 2 shows the measured latency:



Please note that the plot is logarithmic.

Latency grows very quickly as N increases for the non-local runs, whereas the latency is relatively steady for $N \leq 32$ for the local runs, before it, too, begings to grow linearly in N (the measured latencies grow exponentially, but so do the values of N).

Interestingly, but expectedly, latency is lower the smaller N becomes (with only few exceptions, which may or may not be due to errors in measurement); this is of course due to the lessened traffic.

1.4 Reliability of results

The experiments are *not* at all exhausting of the actual performance of the book store. Only one type of randomly generated books were used, and only one set of workload configuration parameters were used. I used parameters that would mostly produce rather *small* interactions. For example, customers only ever buy 3 book copies at a time, and stock managers only ever add 4 new books at a time, and so on. Ideally, I would also have run a test with a large number of books manipulated per interaction.
