Final Project - Transactions

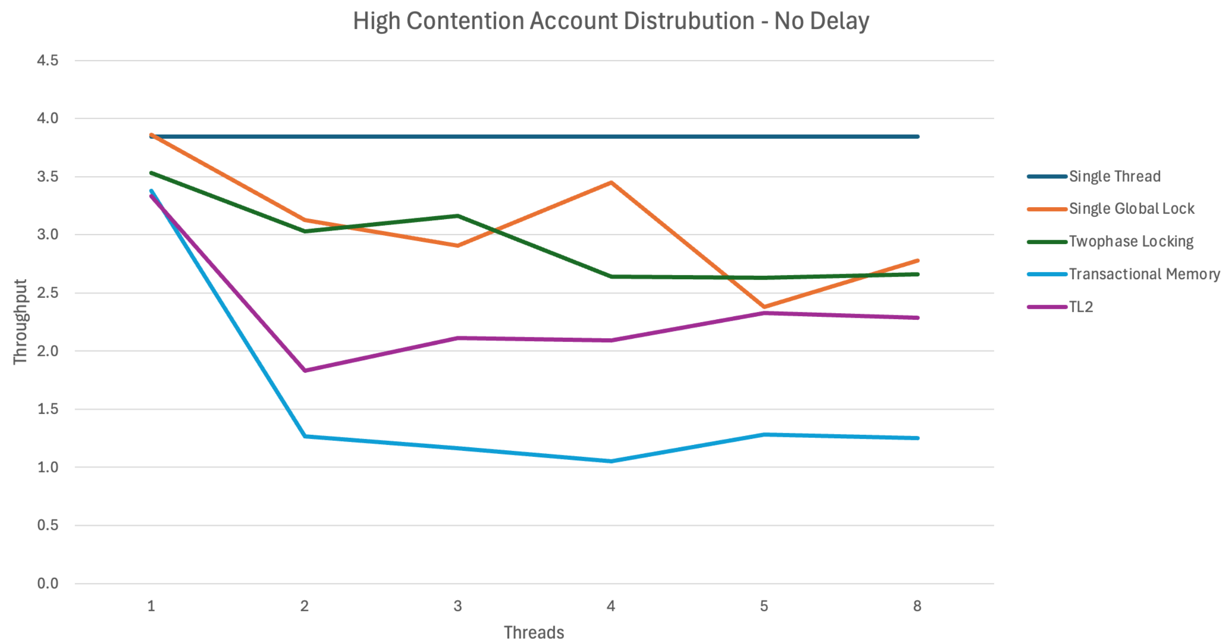
Jasey Chanders

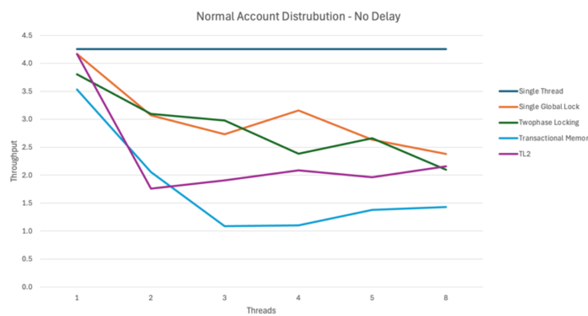
Description

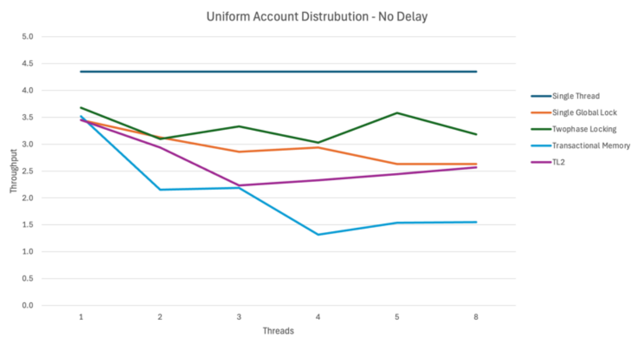
This code is a demonstration of different synchronization strategies for executing large number of transactions across bank accounts. The requirement in this scenario is that any transaction that is executed must be executed in full and not interrupted to prevent corruption of the values of accounts. This is done in 5 different ways: Single Thread (for baseline), Single Global Lock, Twophase Locking, C++ Transactional Memory, TL2 Optimistic Concurrency Control.

Experimental Results

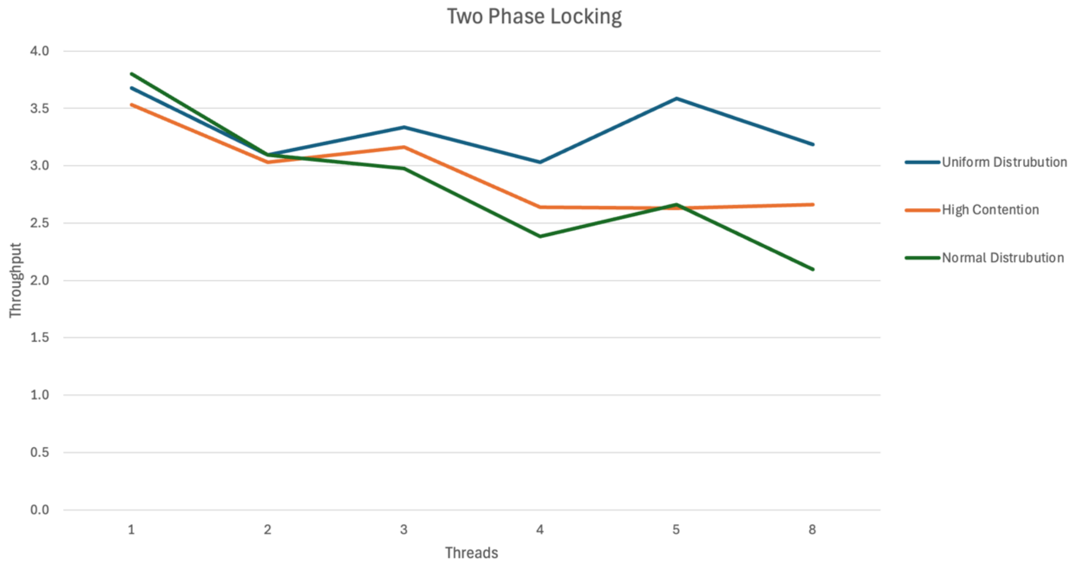
In order to test my synchronization strategies I tested each strategy on three test files with a randomized mixture of transactions and balance checks. Each file contained 1000 accounts and 1 Million transactions/balance checks. The first used a uniform distribution across the accounts, the second used a normal distribution and the third used a normal distribution with a lower standard deviation. Because the actual transaction takes almost no time you can see in the graphs below that every synchronization strategy adds additional overhead and does not improve throughput. So if the transactions are truly this light weight then it makes the most sense to not attempt multi threading. Transactional memory had the highest overhead and thus performed the worst.



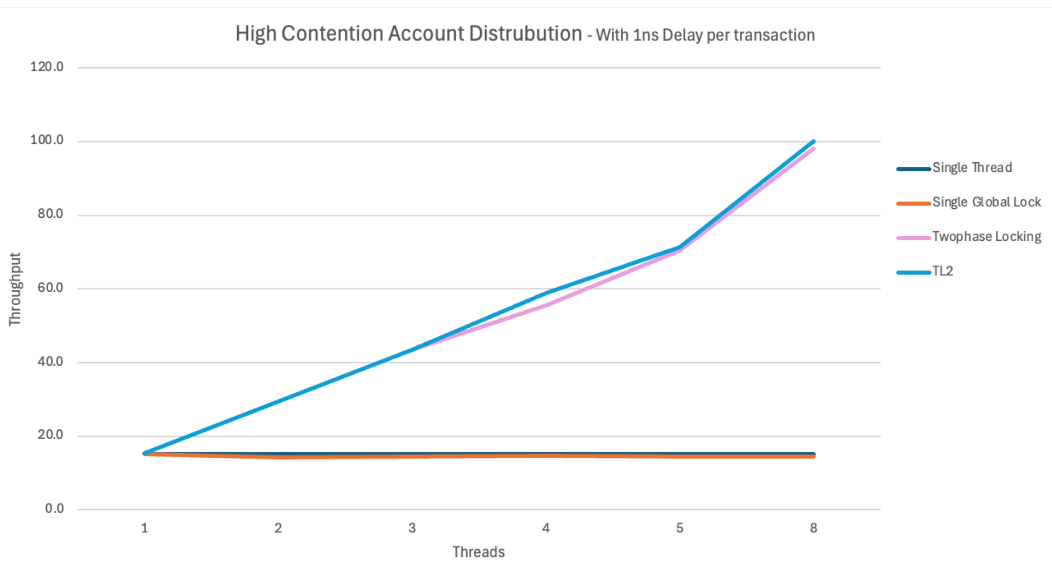


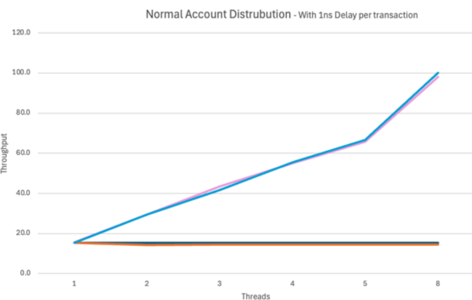


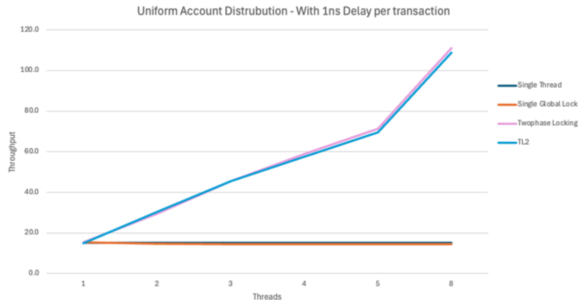
Below can be seen how two phase locking performed on different levels of contention. Uniform distribution has a somewhat higher throughput as expected however the normal distribution and the high contention are fairly similar. I believe this is because the level of contention in the two files ended up being relatively similar.



Because the lightweight transactions do not illustrate the advantage of multithreading I added a 1 nanosecond delay to each transaction to simulate a transaction being a little bit more work. This allows the difference between strategies to be much more obvious. Single Threading and the Single Global Lock perform about the same across any number of threads (for sgl) as even with more threads sgl is functionally single threaded. However tl2 and two phase locking drastically increase their throughput with more threads. Note: I did not run this test for transactional memory because putting threads to sleep is not a transaction safe action and thus I could not simulate it in the same fashion.







Ultimately, if the transactions are lightweight then there is no need to multi- thread. However when the transactions are more work than multi-threading with either tl2 or two phase locking worked quite well.

Description of Code Organization

The code is organized into 3 main functions. The first is an IO processing file that handles reading input arguments, parsing the input files and writing to the output files. It is single threaded and should result in an identical time addition to all tests. The next part of note is my test generation file. This allows for the generation of large data files in the correct format. Finally almost all the work in done in the transactions file. The transactions file first gets the input from IO then it splits into a if statement that effectively separates each style of synchronization completely to allow for accurate timing. Each section of the if statement with the exception of the single threaded case creates any mutexes or barriers it needs then launches the desired number of threads with their corresponding working functions. Once the work is complete they clean up the threads, merge the final output and send it to IO to be written to the output file. Inside each worker function is where the synchronization method is implemented. They distribute work but striding through the vector of input
actions offset by their thread id. Test files are laid out with the starting values of each account first then a randomly generated set of transactions and balance checks. In order to test correctness I ran the single threaded version on each input file and used it as the
correct output then each multi-threaded output file was compared to that using the same autograding method we used for our labs. Because balance checks can happen at anytime depending on which thread executes them, I do not check of accuracy of balance checks but the values are outputted to a separate file.

Files

- IO.cpp/IO.h Parses command line args, reads input files, writes to output files, single threaded
- Transactions.cpp/h handles all synchronizations methods and processes all transactions
- test_file_gen.cpp generates random test files with the needed format for the program
- autograde.sh runs test using a combination of all sync strategies, 1 2 5 8 threads and each input file
- Make file build the transaction program (not the test file program)
- .pngs are all graphs for the write up
- autograde_tests/* are all the test and correct answer files

Compilation and Execution Instructions

To run all tests:
make

To run a specific test:

```
make
```

```
./transactions -i <input-file.txt> -o a<output-file> -t <NUM_THREADS> --strat=<single, sgl, twophase, trmem, tl2>
```

To generate a test file

```
g++ test_file_gen.cpp -o gen
```

```
./gen -a <NUM_ACCOUNTS> -t <NUM_TRANSACTIONS> -o <output-file.txt> -c <-1:uniform-dist, 1:normal-dist, 2:highc>
--max=<largest-transaction-val>
```

Bugs

No known bugs