Concurrent Distributed Dynamic Sized Worker Queue

Yunjia Wang (yunjiaw@andrew.cmu.edu)
Jiaan Dai (jiaand@andrew.cmu.edu)

Oct 30, 2019

URL

`https://flowersh0026.github.io/618-final/`

Summary

We plan to start with implementing a concurrent dynamic sized queue in 4 different synchronization methods and evaluating their performance. The four versions include: coarse-grained locked queue, fine-grained locked queue, lock-free queue using CAS instructions, lock-free queue using transactional memory. Then we would further develop the concurrent queues into distributed versions, and do further performance evaluation there.

Background

Dynamic sized queue is a commonly used basic data structure. A concurrent multiple producer multiple consumer queue can be useful in some parallel scenarios such as work distribution queue. We want to simulate a worker queue in this project.

There are three base operations for a FIFO queue including push, pop and front. All of the base operations can be done in O(1). Push and pop would modify the queue by inserting/removing elements to/from tail/head, while top will only do a read operation on the head. Since we are simulating a worker queue, in addition to the normal base operations, we will have another blocking pop. When calling a blocking pop, thread will be blocked on an empty queue.

In addition to a centralized queue, under scenarios that the queue needs to serve large amount of works, distributed queue could help a lot when single machine could not hold all the data and could also accelerate performance of locked version queues. Therefore, we also want to evaluate the performance of distributed queues and possibly explore different load balancing policies.

Challenge

It can be inherently challenging to implement concurrent data structures due to the data races issues. Although the locked versions can be relatively easy to accomplish, the lock-free versions need some efforts to maintain correctness of the algorithm. For transactional memory, we have extra constraints and need to use special instructions to implement the features.

Distributed version can be inherently difficult as we need to maintain the data consistency and the correct ordering among different machines. When an operation taken place, the metadata on each machine, as well as where the head points to, need to be updated accordingly. Additionally, we also need to explore the load balancing policies under various work loads to let it maintain a reasonable performance.

Resources

We will write code from scratch but using some paper as references, as listed below:

- An Optimistic Approach to Lock-Free FIFO Queues

We will use the GHC cluster for benchmarking our concurrent queues. For the benchmarks in the distributed environment, we will use the latedays cluster. To implement the transactional memory version, we may need an Intel processor that supports the Transactional Synchronization Extensions (TSX).

Goals and Deliverables

Goals

Plan to achieve

- Coarse-grained locked queue
- Fine-grained locked queue
- Lock-free queue using CAS instructions
- Lock-free queue using transactional memory
- Performance analysis

Hope to achieve

- Distributed version of coarse-grained locked queue and fine-grained lock queue
- Distributed version of lock-free queues

Fallback Plan

If the work goes slowly, we will try to finish the first 3 versions listed in the *Plan to achieve* list as well as the performance analysis.

Deliverables

We will show the speedup graphs and the comparisons between different versions of concurrent queues we implement. To demonstrate we did a good job, we will provide efficient implementations of lock-free concurrent queues as well as the performance analysis to show that when we should use a lock-free version to achieve higher efficiency.

Our project is a system project, so the main goal is to implement a better concurrent queue using lock-free techniques. However, because of the nature between the optimistic and pessimistic concurrency control, there should be some scenarios that it is inappropriate to adopt a lock-free queue. Therefore, we will also provide an analysis about it.

Platform Choice

We will mainly use the GHC machines as our development platform, and we will use C++ as the main programming language. C++ provides good zero-cost abstractions that allow us to implement complicated data structures with less efforts, and it also gives us full control on the platform and runtime. So we believe it is a good choice to implement the concurrent queues.

Schedule

Week	Task
Nov 4 - Nov 10	Understand the data structure of lock-free queue using CAS
	Understand the data structure of lock-free queue using transactional memory
Nov 11 - Nov 17	Implement the lock-free queue using CAS
	Implement the lock-free queue using transactional memory
Nov 18 - Nov 24	Implement the locked versions (coarse-grained and fine-grained)
	Explore the implementations of distributed concurrent queues
Nov 25 - Dec 1	Implement the distributed versions
Dec 2 - Dec 8	Performance analysis and reports