Data Structure Research Paper

BQ: A Lock-Free Queue with Batching

John Mirschel and Timothy Rigby

March 2019

Abstract

This paper describes an implementation of a lock-free queue that utilizes batching to increase performance by up to 16x over standard lock-free queues. We attempted to implement the design and algorithms from the original research paper BQ: A Lock-Free Queue with Batching and tried to reproduce similar functionality and performance. The original implementation of this data structure was done in C++, but we wanted to try and build this in Java so we could compare the results of how the data structure performs in two different languages.

1 Introduction

Our implementation of the lock-free queue provides all of the functionality available to a sequential queue. Our lock free batching extension builds upon the simple concurrent lock free queue implemented by Michael and Scott by vastly improving on scalability of the concurrent lock free queue. A concurrent queue has two access points of high contention, the head of the queue from which nodes are dequeued and the tail of the queue into where new nodes are enqueued. This inherently limits throughput of the data structure as only one thread can enqueue and one thread can dequeue at a time. Our version of a lock-free queue uses the idea of the future programming construct to perform batching operations as first seen in the paper written by Kogan and Herlithy[1]. Batching means to just group a sequence of standard operations to just one single batch operation, which then applies them together to the shared data structure.

Kogan and Herlithy originally proposed that operations with the same type (enqueue or dequeue) get added to the shared queue all at once. This means they execute each subsequence of enqueues together by appending them in order to the tail of the queue, and each subsequence of dequeues is removed from the head of the list all at the same time. The problem with this implementation is that performance can degrade if the operations in the batch frequently switch between enqueue and dequeue, as their implementation only allows for a batch to be built from sequential and like operations. The algorithm we implemented improves on this idea by batching these operations locally. Local batching allows our algorithm to handle both enqueue and dequeue operations in any order and build the result before applying it to the shared queue. This means it applies the batch operation all at once to the shared queue to reduce contention between multiple threads. This helps solve the problem that concurrent queues have with the contention of the head and tail pointers by batching these operations locally, which reduces the number of access attempts to the shared queue and improves scalability.

2 Related Works

The Lock Free Queue with Batching designed by Gal Milman et al. [2] is an extension of the concurrent lock-free queue by Michael and Scott [3]. The Michael and Scott queue serves as the baseline for many developments on the concurrent FIFO queue, yet it is not scalable and only allows for one Enqueuer and one Dequeuer thread at a time. Moir et al. [4] presented a queue that increased scalability by allowing pairs of concurrent equeue and dequeue method calls to exchange values and eliminate themselves without having to access the shared queue under certain circumstances. This concept reduces contention upon the shared queue, a concept adopted and furthered by the Lock Free Queue with Batching as it locally pairs and eliminates multiple enqueue and dequeue operations before accessing the shared queue. The implementation of the Lock Free Queue with Batching most directly rises from the work done by Kogan and Herlihy which allows for batching of operations of the same type to be executed as a subsequence at once on the shared queue. The Lock Free Queue with Batching improves on the performance of the concurrent queue implemented by Kogan and Herlihy by allowing for alternation between enqueue and dequeue operations within a batch such as would most often be encountered in the general use case of a

queue.

3 Overview

The Lock Free Queue with Batching is a direct extension of the basic concurrent Lock Free Queue. Our first step in building this data structure is to begin with a well-performing lock free queue implementation. The Lock Free Queue with Batching allows for quick computation of its size by maintaining counters with the head and tail of the shared queue. Batching threads will receive enqueue and dequeue operations to execute locally, building a subsequence of a queue that can be then applied to the shared queue atomically to perform multiple enqueue and dequeue operations in a single access to the shared queue. Local threads will also track the number of dequeue operations taking place to be able to quickly modify the shared queue as needed while applying the batch. When a batch is applied to the shared queue, a descriptor like object replaces the head of the queue in order to have all other threads assist the completion of the batching event.

References

[1] Alex Kogan and Maurice Herlihy. "The future(s) of shared data structures". In: *PODC* (2014).

[2] Gal Milman, Victor Luchangco, Alex Kogan, et al. "BQ: A Lock-Free Queue with Batching". In: *SPAA* (July 2018).

[3] Maged M. Michael and Michael L. Scott. "Simple, fast, and practical non-blocking and blocking concurrent queue algorithms". In: *PODC* (1996).

[4] Mark Moir, Daniel Nussbaum, Ori Shalev, et al. "Using elimination to implement scalable and lock-free FIFO queues". In: *SPAA* (2005).