Principles and Practice of Parallel Programming 2021 (PPoPP '21)

POSTER: A Lock-free Relaxed Concurrent Queue for Fast Work Distribution

Giorgos Kappes, Stergios V. Anastasiadis
University of Ioannina, Ioannina 45110, Greece

The Producer-Consumer Problem

Appears in multicore machines

- Kernel & user-level I/O stacks, networking, parallel applications
- Requires fast communication at high concurrency

Communication

- Shared data structure often implemented as concurrent FIFO queue
- Multiple producer & consumer threads access the queue
- The item with longest time in the queue is removed first

Issues

- Time-based item ordering (e.g., FIFO) inherently sequential
- Limits concurrency, delays queue operations

Related Work

Concurrent FIFO queues

- Lock-free Michael-Scott queue: CAS retries limit performance
- Lock-free LCRQ: Operation retries & queue creations limit performance
- Wait-Free Queue: Performance sensitivity to frequency of slow path relatively to fast
- Blocking Broker Queue: Slower at empty or full queue

Relaxed queues and bags

- Multilane (concurrent multiset): No wait fairness
- Priority relaxation: Item priority probabilistically relaxed
- Lock-based RCQB: Locking limits concurrency

Relaxed Concurrent Queue Single (RCQS)

Goals

- High operation throughput
- Low item wait latency in the queue

Relaxed ordering model: split enqueues/dequeues into 2 stages

- 1st Stage: Distribute operations sequentially
- 2nd Stage: Let them complete concurrently, potentially out of order

Implementation

- Fixed-size (power of 2) circular buffer (slots)
- Head and Tail unsigned integers
- Slot state (occupied/free, 1-bit value), slot data (63-bit value)
- Slot state is also a condition variable that notifies waiting dequeuers

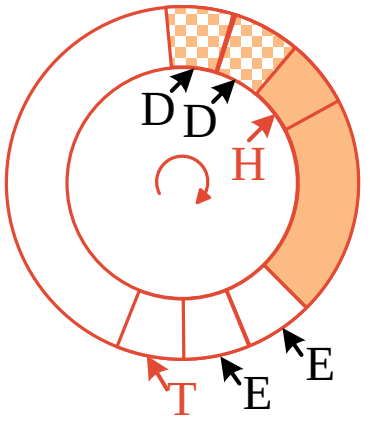
Operations of RCQS

Enqueue

- Allocate slot to thread sequentially with FAA on tail
- Single Compare-and-Swap (CAS): switch slot state from free to occupied and update slot data atomically
- If the CAS succeeds, enqueue completes & notifies waiting dequeuers (if any), otherwise retry at same slot

Dequeue

- Allocate slot to thread sequentially with FAA on head
- Single Compare-and-Swap (CAS): switch slot state from occupied to free and zero out slot data atomically
- If the CAS succeeds, dequeue returns retrieved data
- If the CAS fails & the slot state is occupied, retry at same slot
- If the slot state is free, sleep after max number of retries



Dequeuers follow the enqueuers

Properties of RCQS

Lock-free: some operation always finishes in a finite number of steps

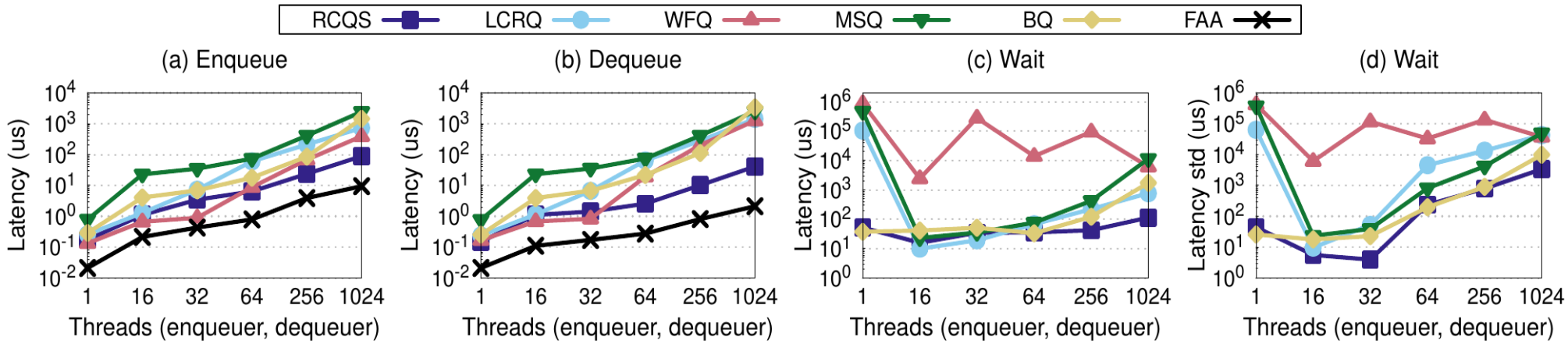
- Assume partial definition to satisfy non-blocking when enqueuer or dequeuer waits on slot for free or occupied precondition
- When a competing thread completes the CAS on a slot, an operation finishes in a finite number of steps
- From sequential slot allocation, enqueuers cannot indefinitely wait at a slot if there are active dequeuers (and vice versa)

Linearizability: the structure remains valid and operations finish between invocation and response

- 2 Invariants: Sequential operation assignment to slot & safe item removal satisfied by FAA and CAS respectively
- For each operation assume two methods with distinct invocation-response: slot allocation and slot update
- Each method has its own linearization point (FAA or CAS)

Evaluation (Dual 16C/32HT Xeon 5218)

Open System with separate threads for enqueuer & dequeuer



RCQS achieves substantially lower operation & wait latency

- **Enqueue latency (lower):** e.g., up to 27.7x than MSQ
- **Dequeue latency (lower):** e.g., up to 34.9x than LCRQ
- **Wait latency (lower & stable):** RCQS avg 16-114μs and std up to 3.4ms
- Wait latency of LCRQ, WFQ reaches tens or hundreds of milliseconds
- Issues: Operation retries for FIFO, queue creations, thread helping

Conclusions & Future Work

Queues with typical queueing discipline (e.g., FIFO)

- Time-based item ordering inherently sequential
- Limit concurrency and delay operations

Relaxed Concurrent Queue Single

- Linearizable, lock-free queue algorithm that relaxes operation ordering
- (1) Sequentially assign operations to slots, (2) execute them concurrently

Benefits

- Utilizes limited amount of memory space
- Improves concurrency, decreases operation & item wait latency

Future Work

- Formal study of the queue properties
- Further experimental evaluation across different system configurations