Brief Announcement: Scalability versus Semantics of Concurrent FIFO Queues[*]

Christoph M. Kirsch Hannes Payer Harald Röck Ana Sokolova
Department of Computer Sciences
University of Salzburg, Austria
firstname.lastname@cs.uni-salzburg.at

ABSTRACT

Maintaining data structure semantics of concurrent queues such as first-in first-out (FIFO) ordering requires expensive synchronization mechanisms which limit scalability. However, deviating from the original semantics of a given data structure may allow for a higher degree of scalability and yet be tolerated by many concurrent applications. We introduce the notion of a k-FIFO queue which may be out of FIFO order up to a constant k (called semantical deviation). Implementations of k-FIFO queues may be distributed and therefore be accessed unsynchronized while still being starvation-free. We show that k-FIFO queues whose implementations are based on state-of-the-art FIFO queues, which typically do not scale under high contention, provide scalability. Moreover, probabilistic versions of k-FIFO queues improve scalability further but only bound semantical deviation with high probability.

Categories and Subject Descriptors

D.1.3 [**Software**]: Concurrent Programming

General Terms

Algorithms, Performance

1. INTRODUCTION

The scalability of applications is limited by Amdahl's Law, which states that the degree to which we can speed up an application on a multi-core system is limited by the amount of code that cannot be parallelized and must be executed sequentially. Since operations on shared data structures may not be fully parallelized there is an intrinsic concurrency bottleneck in many applications using shared data structures that gets increasingly problematic with an increasing number of cores.

Even basic concurrent data structures such as stacks and queues have negative scalability under high contention due to synchronization. However, maintaining data structure semantics may actually not be needed in some concurrent applications. Consider, for example, a webserver which stores incoming requests in a shared FIFO queue running on a server machine with possibly hundreds of cores. The requests are dequeued and handled by worker threads at a later point in time. In such a scenario it may not be important to process the requests in perfect FIFO order. Instead, it may be sufficiently fair to handle the requests FIFO up to a constant that bounds the deviation from FIFO order. A detailed evaluation of the trade-off between scalability and semantical deviation of concurrent data structures can be found in [3]. A more recent survey exemplifies the trend towards scalable but semantically weaker concurrent data structures [5].

[*]Supported by the EU ArtistDesign Network of Excellence on Embedded Systems Design, the National Research Network RiSE on Rigorous Systems Engineering (Austrian Science Fund S11404-N23), and an Elise Richter Fellowship (Austrian Science Fund V00125).

2. K-FIFO QUEUE

A k-FIFO queue provides an enqueue and a dequeue operation similar to a regular FIFO queue. Logically, a k-FIFO queue is a queue where an enqueue operation adds an element to the queue tail and a dequeue operation removes one of the $k-e$ oldest elements from the queue with e being the number of dequeue operations since the most recent dequeue operation that removed the oldest element from the queue, i.e., $e < k$ always holds. Thus, retrieving the oldest element from the queue may require up to k dequeue operations, which may not return any element younger than the k oldest elements in the queue and which may be interleaved with any number of enqueue operations. This implies that k-FIFO queues are starvation-free. Note that a 0-FIFO queue is equivalent to a regular FIFO queue.

We implement a k-FIFO queue using p versions of a regular FIFO queue, so-called partial FIFO queues, and a load balancer that distributes data structure operations among the p partial FIFO queues [3]. The value of p and the type of load balancer determine k (semantical deviation), as discussed below, as well as the scalability of the k-FIFO queue, i.e., how many data structure operations can potentially be performed concurrently and in parallel without causing contention. Note that p and the load balancer can be configured by the programmer at compile time or dynamically at runtime with the help of performance counters. For example, a load balancer may be chosen with $p = 1$ under low contention and with increasing p as contention increases.

A metric for the quality of the load balancer is the maximum imbalance of operations of a given operation type, which we define as the difference between the partial FIFO queue on which the most and the fewest operations of a given type have been performed at a given point in time. Starvation of elements in a k-FIFO queue is prevented if the maximum imbalance of operations for each operation type is bounded. In this case the semantical deviation of a k-FIFO queue from a regular FIFO queue is also bounded.

A load balancer that provides bounded semantical deviation for a k-FIFO queue can be implemented with two global counters indicating on which partial FIFO queue the last enqueue and the last dequeue operation was performed. We refer to it as perfect load balancer. The global counters are accessed and modified using atomic operations, which can cause cache conflicts when multiple threads try to modify the same memory locations concurrently. However, scalability may still be achieved under low concurrent load since the load balancer itself is simple and contention on the shared memory locations may rarely happen. It can be shown that if t threads perform concurrent data structure operations on a k-FIFO queue using the perfect load balancer the semantical deviation is $k \leq t \cdot (p-1)$.

A load balancer that randomly distributes operations over the partial FIFO queues bounds the semantical deviation probabilistically. This approach, also known as randomized load balancing [2], has been shown to provide good distribution quality if the random numbers are distributed independently and uniformly. However, generating such random numbers may be computationally expensive. Therefore, it is essential to find the right trade-off between quality and overhead of random number generation. Suppose that m enqueue operations and n dequeue operations have been performed on p partial FIFO queues using a random load balancer, then it can be shown that with high probability the semantical deviation is bounded by $k \leq \Theta\left(\sqrt{\frac{(m+n)\log p}{p}}\right) \cdot (p-1)$.

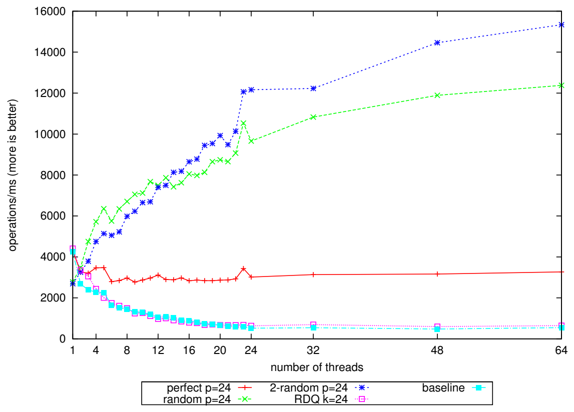
In order to improve the balancing quality of the random load balancer, d partial FIFO queues with $1 < d \leq p$ may be chosen randomly. Out of the d partial FIFO queues the queue that contributes most to a better balance is then selected. For example, an enqueue operation may be performed on the partial FIFO queue that contains the fewest elements. We refer to such a load balancer as d-random load balancer. The runtime overhead of the d-random load balancer increases linearly in d since the random number generator is called d times. It can be shown that a d-random load balancer bounds the semantical deviation with high probability to $k \leq 2 \cdot \Theta\left(\frac{\log\log p}{d}\right) \cdot (p-1)$.

3. EVALUATION

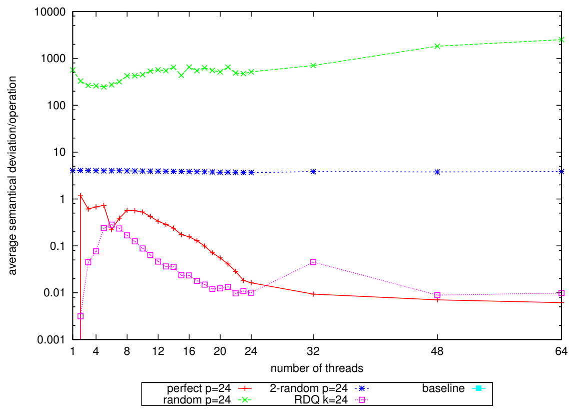
The experiments ran on a 24-core server machine (four 6-core 2.1GHz AMD Opteron processors). The benchmarks start with empty queues. Then, each thread enqueues and dequeues elements in alternating order. We use the number of operations performed by all threads per millisecond as our metric of throughput. Additionally, we compute from benchmark traces the average semantical deviation per operation of the different queues. We compare our k-FIFO queues with the lock-free Michael-Scott FIFO queue [4] (baseline) and a modified version of that FIFO queue called Random Dequeue Queue (RDQ) [1]. RDQ is semantically equivalent to a k-FIFO queue where k can be configured at compile time. The implementation of our k-FIFO queue uses the previously described load balancers and the Michael-Scott FIFO queue for the partial FIFO queues.

The throughput results are depicted in Figure 1(a). The throughput of the Michael-Scott baseline and RDQ decreases with an increasing number of threads. The k-FIFO queue with the perfect load balancer does not scale but performs better than the baseline. The k-FIFO queues with the random and the 2-random load balancers provide scalability.

The semantical deviation results are depicted in Figure 1(b). The average semantical deviation of the k-FIFO queue with the random load balancer is high. In comparison, the average semantical deviation of the k-FIFO queue with the 2-random load balancer is three orders of magnitude lower. The lowest average semantical deviation is achieved by RDQ and the k-FIFO queue with the perfect load balancer. Note that the semantical deviation of the Michael-Scott baseline is zero.



(a) Throughput



(b) Semantical Deviation

Figure 1: Increasing number of threads on a 24-core server machine

The k-FIFO queue with the 2-random load balancer provides the best compromise between scalability and semantical deviation for the presented workload. Depending on the workload a k-FIFO queue may be configured with the smallest p value and the most accurate load balancer to provide the best adherence to data structure semantics while still providing scalability. Interesting future work includes applying our implementation concept to other concurrent data structures and transaction-based systems such as software-transactional memory.

4. REFERENCES

[1] Y. Afek, G. Korland, and E. Yanovsky. Quasi-linearizability: Relaxed consistency for improved concurrency. In *Proc. Conference on Principles of Distributed Systems (OPODIS)*, pages 395–410. Springer, 2010.

[2] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking. Balanced allocations: The heavily loaded case. *SIAM Journal on Computing*, 35(6):1350–1385, 2006.

[3] C. Kirsch, H. Payer, and H. Röck. Scal☠: Non-linearizable computing breaks the scalability barrier. Technical Report 2010-07, Department of Computer Sciences, University of Salzburg, November 2010.

[4] M. Michael and M. Scott. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In *Proc. Symposium on Principles of Distributed Computing (PODC)*, pages 267–275. ACM, 1996.

[5] N. Shavit. Data structures in the multicore age. *Communications of the ACM*, 54:76–84, March 2011.