How FIFO is Your Concurrent FIFO Queue?

Andreas Haas, Christoph M. Kirsch, Michael Lippautz, Hannes Payer

University of Salzburg

RACES Workshop, October 2012
Strict vs. Relaxed FIFO Queues

- **Strict FIFO queue implementations**: linearizable with respect to strict FIFO queue semantics
- **Relaxed FIFO queue implementations**
Strict vs. Relaxed FIFO Queues

strict FIFO queue implementations
  linearizable with respect to strict FIFO queue semantics

relaxed FIFO queue implementations
  linearizable with respect to relaxed FIFO queue semantics
Strict vs. Relaxed FIFO Queues

- **Strict FIFO queue implementations**
  - linearizable with respect to strict FIFO queue semantics

- **Relaxed FIFO queue implementations**
  - linearizable with respect to relaxed FIFO queue semantics

- bounded out-of-order treatment of queue elements possible
Strict vs. Relaxed FIFO Queues

Strict vs. Relaxed FIFO Queues

Strict FIFO queue implementations

linearizable with respect to strict FIFO queue semantics

Relaxed FIFO queue implementations

linearizable with respect to relaxed FIFO queue semantics

bounded out-of-order treatment of queue elements possible
Strict vs. Relaxed FIFO Queues

- **Strict FIFO queue implementations**
  - linearizable with respect to strict FIFO queue semantics

- **Relaxed FIFO queue implementations**
  - linearizable with respect to relaxed FIFO queue semantics

- Bounded out-of-order treatment of queue elements possible
Strict vs. Relaxed FIFO Queues

strict FIFO queue implementations

linearizable with respect to strict FIFO queue semantics

relaxed FIFO queue implementations

linearizable with respect to relaxed FIFO queue semantics

bounded out-of-order treatment of queue elements possible
How FIFO are Relaxed FIFO Queues?

- Some people say relaxed FIFO queues are not enough FIFO.
  - No applications for relaxed FIFO queues.
How FIFO are Relaxed FIFO Queues?

- Some people say relaxed FIFO queues are not enough FIFO.
  - No applications for relaxed FIFO queues.

We say relaxed FIFO queue implementations can be even more FIFO than strict FIFO queue implementations.
Example
Example

linearization point

enq a

enq b

deq b

deq a
Example

linearization point

\[ \text{enq} \quad a \quad \text{enq} \quad b \]

\[ \text{deq} \quad b \quad \text{deq} \quad a \]

linearizable
Example

linearization point

enq a

enq b

deq b

deq a

linearizable
Example

linearization point

![Timeline with enq and deq operations](image)

- **Linearizable** case:
  - Time: a, b
  - Linearization point: b
  - Result: linearizable

- **Not linearizable** case:
  - Time: a, b
  - Linearization point: a
  - Result: not linearizable
Example

Linearization point

Out-of-order execution of overlapping operations

Out-of-order treatment of queue elements

Linearizable

Not linearizable
Key Idea

1.) Record concurrent histories of various FIFO queue implementations.

2.) Analyze these concurrent histories using only the invocation times of operations.
1.) Record concurrent histories of various FIFO queue implementations.

2.) Analyze these concurrent histories using only the invocation times of operations.
   - Ideally operations would take zero time
Key Idea

1.) Record concurrent histories of various FIFO queue implementations.

2.) Analyze these concurrent histories using only the invocation times of operations.
   ▶ Ideally operations would take zero time
   ▶ Independent of the execution time of operations
Element-Fairness

enq a

enq b

deq b

deq a
Element-Fairness

enq a  enq b  deq b  deq a

zero-time linearization
Element-Fairness

Element: $b$ overtakes element $a$

Enq $a$, Enq $b$, Deq $b$, Deq $a$

Zero-time linearization
Element-Fairness

**Definition**

\[\text{element-fairness} = \text{number of overtakings in the zero-time linearization}\]
Experiments

all threads do in parallel

for 10,000 iterations
{
    enqueue unique element

dequeue element
}
Experiments

all threads do in parallel

for 10,000 iterations
{
    enqueue unique element
    calculate Pi
    dequeue element
    calculate Pi
}
Experiments

all threads do in parallel

for 10,000 iterations
{
    enqueue unique element
    calculate Pi
    dequeue element
    calculate Pi
}

- No dequeues in the first 200 iterations to avoid empty checks.
- No enqueues in the last 200 iterations to empty the queue.
Element-Fairness per Element

80 threads

element-fairness per element (logscale, less is better)

computational load (logscale)
Element-Fairness per Element

80 threads

Element-fairness per element (logscale, less is better)

Computational load (logscale)

- LB
- MS
- FC

8/17
Element-Fairness per Element

80 threads

computational load (logscale)

element-fairness per element (logscale, less is better)

LB
MS
FC
US k-FIFO (k=80)
Element-Fairness per Element

80 threads

element-fairness per element (logscale, less is better)

computational load (logscale)

LB
MS
FC
US k-FIFO (k=80)
RR Scal (p=80)
2RA Scal (p=80)
Operation-Fairness

- Measure the out-of order execution of single operations
Operation-Fairness

- Measure the out-of order execution of single operations
- Observation: Linearization points induce a strict order on the queue operations
Operation-Fairness

- Measure the out-of order execution of single operations
- Observation: Linearization points induce a strict order on the queue operations
Operation-Fairness

- Measure the out-of order execution of single operations
- Observation: Linearization points induce a strict order on the queue operations
operation enq \( b \) overtakes operation enq \( a \)
operation enq \( b \) overtakes operation enq \( a \)

Definition

operation-fairness = number of overtakings in a concurrent history
Operation-Age and Operation-Lateness

**Definition**

\[
\text{age (op)} = \text{number of operations op overtakes}
\]
operation-age and operation-lateness

Definition

\[ \text{age (op)} = \text{number of operations op overtakes} \]

\[ \text{age(enq } a\text{)} = 0 \]
\[ \text{age(enq } b\text{)} = 1 \]
Operation-Age and Operation-Lateness

**Definition**

\[ \text{age (op)} = \text{number of operations op overtakes} \]

**Definition**

\[ \text{lateness (op)} = \text{number of operations which overtake op} \]

- \( \text{age(enq \ a)} = 0 \)
- \( \text{age(enq \ b)} = 1 \)
**Operation-Age and Operation-Lateness**

**Definition**

age (op) = number of operations op overtakes

**Definition**

lateness (op) = number of operations which overtake op

- age(enq a) = 0
- age(enq b) = 1
- lateness(enq a) = 1
- lateness(enq b) = 0
Experiments (2)

- Only for strict FIFO queue implementations at the moment.
- Measuring relaxed implementations is future work.
Experiments (2)

- Only for strict FIFO queue implementations at the moment.
- Measuring relaxed implementations is future work.

All threads do in parallel:

for 10,000 iterations
{
    enqueue unique element
    calculate Pi
}
Experiments (2)

- Only for strict FIFO queue implementations at the moment.
- Measuring relaxed implementations is future work.

```plaintext
for 10,000 iterations
{
    enqueue unique element
    calculate Pi
}
```

All threads do in parallel

One thread does

dequeue all elements sequentially
Maximum Operation-Age

80 threads

maximum operation-age (logscale, less is better)

computational load (logscale)

LB  MS  FC
Maximum Operation-Lateness

80 threads

![Graph showing computational load vs. maximum operation-lateness for different algorithms. The x-axis represents computational load on a logarithmic scale, and the y-axis represents maximum operation-lateness also on a logarithmic scale. The graph compares four algorithms: LB, MS, FC, and an unspecified one represented by a blue line. Each line shows a different trend under varying computational loads.](image-url)
Number of Overtaking Operations

![Graph showing the number of enqueue operations with operation-age > 0 for different computational loads. The graph compares different algorithms: LB, MS, FC, and an unspecified algorithm (indicated by stars). The x-axis represents computational load on a log scale, and the y-axis shows the percentage of enqueue operations with operation-age > 0. The graph illustrates how the percentage decreases as computational load increases.]
Number of Overtaken Operations

% of enqueue operations with operation-lateness > 0

(less is better)

computational load (logscale)

80 threads

LB, MS, FC
Conclusion

- We introduced metrics to compare the behavior of various FIFO queue implementations.
  - Relaxed implementation can appear more FIFO than strict implementations.

Future work
- Measure operation-fairness of relaxed FIFO queue implementations.
- Use element-fairness to analyze implementation of other data structures, e.g. stacks.
Thank You

For more information about the queue implementations see http://scal.cs.uni-salzburg.at/

Additional measurement results can be seen on http://scal.cs.uni-salzburg.at/races12/