Scalability versus Semantics of Concurrent Objects

Christoph M. Kirsch, Hannes Payer, Harald Röck, Ana Sokolova

University of Salzburg

WRiSE 2011
Motivation

• Applications consist of concurrent objects
• Concurrent object semantics is determined by
  • Sequential specification (e.g. FIFO)
  • Consistency condition (e.g. linearizability)
• Implementation of concurrent object semantics affect performance and scalability of applications
Motivation - Scalability

throughput

number of cores
Motivation - Scalability

Throughput vs. Number of Cores

Linear scalability
Motivation - Scalability

- Linear scalability
- Positive scalability

-throughput vs. number of cores
Motivation - Scalability

Throughput vs. Number of Cores

- Linear scalability
- Positive scalability
- Negative scalability
Motivation - Scalability

Throughput vs. number of cores:
- Linear scalability
- Positive scalability
- Negative scalability
- Low performance
Motivation - Scalability

- Linear scalability
- Positive scalability (high performance)
- Negative scalability
- Positive scalability (low performance)

throughput vs. number of cores
Motivation - Amdahl’s Law

\[
\text{speedup} = \frac{1}{(1 - p) + \frac{p}{n}}
\]

number of cores \( n \)

95%  
90%  
75%  
50%  

%... portion of parallel code \( p \)
Motivation - Parallelization

Sequential
- System
- Concurrent Objects

Parallel
- Concurrent Objects
- System
- Application Logic
Motivation - Parallelization

Sequential
- System
- COs
- Concurrent Objects

Parallel
- System
- Application Logic
Motivation - High Contention

Sequential

Parallel

S COs Concurrent Objects S A
Motivation -
Low Contention

Sequential

Parallel

S  COs  Concurrent Objects  S  Application Logic
Research Question

Can we get high performance and positive scalability for concurrent objects on multi-core systems under high contention?
Yes we can!
Sequential Specification

- Defines the state and operations of the object and how each operation changes the object’s state
- Example FIFO queue:
Sequential Specification

• Defines the state and operations of the object and how each operation changes the object’s state

• Example FIFO queue:
Sequential Specification

- Defines the state and operations of the object and how each operation changes the object’s state
- Example FIFO queue:
Sequential Specification

- Defines the state and operations of the object and how each operation changes the object’s state
- Example FIFO queue:
 Sequential Specification

- Defines the state and operations of the object and how each operation changes the object’s state
- Example FIFO queue:

![Diagram of FIFO queue with nodes labeled 1, 2, 3, 4, and arrows indicating enqueue process at the tail]
Sequential Specification

- Defines the state and operations of the object and how each operation changes the object’s state
- Example FIFO queue:

```
1 2 3 4
```

head .enqueue  tail
Sequential Specification

- Defines the state and operations of the object and how each operation changes the object's state
- Example FIFO queue:
Sequential Specification

- Defines the state and operations of the object and how each operation changes the object’s state
- Example FIFO queue:

```
<table>
<thead>
<tr>
<th>dequeue</th>
<th>head</th>
<th>tail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>
```
Sequential Specification

- Defines the state and operations of the object and how each operation changes the object’s state
- Example FIFO queue:

  dequeue

  ![Diagram of FIFO queue with positions 1, 2, 3, 4, and arrows indicating dequeue, head, and tail]
Sequential Specification

• Defines the state and operations of the object and how each operation changes the object’s state

• Example FIFO queue:

dqueue | head    | tail

1 → 2 → 3 → 4
Sequential Specification

• Defines the state and operations of the object and how each operation changes the object’s state

• Example FIFO queue:
Sequential Specification

- Defines the state and operations of the object and how each operation changes the object’s state
- Example FIFO queue:

```
2 --> 3 --> 4
```

dequeue

head

```
2
```

tail

```
4
```
Consistency Condition

• Determines the order of when operations take effect at a given concurrent object

• Example Linearizability (FIFO queue):

![Diagram of FIFO queue with elements 1, 2, 3, and arrows pointing from head to tail]
Consistency Condition

- Determines the order of when operations take effect at a given concurrent object
- Example Linearizability (FIFO queue):

```
head
  ↓
1 → 2 → 3
  ↓
tail
parallel enqueue
```
Consistency Condition

• Determines the order of when operations take effect at a given concurrent object

• Example Linearizability (FIFO queue):

1  2  3  4
Consistency Condition

- Determines the order of when operations take effect at a given concurrent object
- Example Linearizability (FIFO queue):

```
head
↓
1 → 2 → 3

parallel enqueue
↓
4 → 5
```

tail
Consistency Condition

- Determines the order of when operations take effect at a given concurrent object
- Example Linearizability (FIFO queue):
Consistency Condition

- Determines the order of when operations take effect at a given concurrent object
- Example Linearizability (FIFO queue):

![Diagram showing a FIFO queue with elements 1, 2, 3, 5, 4, and pointers indicating the head and tail positions.]
Consistency Condition

- Determines the order of when operations take effect at a given concurrent object
- Example Linearizability (FIFO queue):

```
head ↓
1 → 2 → 3 → 5
```

tail ↓

4
Consistency Condition

- Determines the order of when operations take effect at a given concurrent object
- Example Linearizability (FIFO queue):
Consistency Condition

- Determines the order of when operations take effect at a given concurrent object
- Example Linearizability (FIFO queue):

```
head
1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 4 \rightarrow tail
```
Consistency Condition

- Determines the order of when operations take effect at a given concurrent object
- Example Linearizability (FIFO queue):
Consistency Condition

- Determines the order of when operations take effect at a given concurrent object
- Example Linearizability (FIFO queue):

```
head
1 -- 2 -- 3 -- 5 -- 4 -- tail
```
Consistency Condition

- Determines the order of when operations take effect at a given concurrent object
- Example Linearizability (FIFO queue):

```
1 → 2 → 3 → 5 → 4
```

head      tail  sequential enqueue
Consistency Condition

• Determines the order of when operations take effect at a given concurrent object

• Example Linearizability (FIFO queue):

```
head
1 ---> 2 ---> 3 ---> 5 ---> 4
          ^
          v
          tail

sequential enqueue

4 ---> 6
```
Consistency Condition

- Determines the order of when operations take effect at a given concurrent object
- Example Linearizability (FIFO queue):

```
head

1 -> 2 -> 3 -> 5 -> 4

<table>
<thead>
<tr>
<th>tail</th>
<th>sequential enqueue</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
```
Consistency Condition

- Determines the order of when operations take effect at a given concurrent object
- Example Linearizability (FIFO queue):

![Diagram of a FIFO queue with sequential enqueue operations from 1 to 7]
Consistency Condition

- Determines the order of when operations take effect at a given concurrent object
- Example Linearizability (FIFO queue):

```
head
1 -> 2 -> 3 -> 5 -> 4 -> 6 -> 7
```

tail
Consistency Condition

- Determines the order of when operations take effect at a given concurrent object
- Example Linearizability (FIFO queue):
Consistency Condition

• Determines the order of when operations take effect at a given concurrent object

• Example Linearizability (FIFO queue):

```
head 1 2 3 5 4 6 tail
```
Consistency Condition

• Determines the order of when operations take effect at a given concurrent object

• Example Linearizability (FIFO queue):

```
| head | 1 | 2 | 3 | 4 | 5 | 6 | 7 | tail |
```
Consistency Condition

- Determines the order of when operations take effect at a given concurrent object
- Example Linearizability (FIFO queue):

```
head -> 1 -> 2 -> 3 -> 5 -> 4 -> 6 -> tail
```

1 2 3 5 4 6 7
Consistency Condition

- Determines the order of when operations take effect at a given concurrent object
- Example Linearizability (FIFO queue):

```
1 → 2 → 3 → 5 → 4 → 6 → 7
```

head → tail
Scalability of Linearizable FIFO Queues Under High Contention

24 cores server machine
Scalability of Linearizable FIFO Queues Under High Contention

24 cores server machine
Scalability of Linearizable FIFO Queues Under High Contention

24 cores server machine
Scalability of Linearizable FIFO Queues Under Low Contention

24 cores server machine
Scalability of Linearizable FIFO Queues Under Low Contention

positive scalability

24 cores server machine
Scalability of Linearizable FIFO Queues Under Low Contention

24 cores server machine
Scalability of Linearizable FIFO Queues Under Low Contention

24 cores server machine

negative scalability
Trade off semantics for performance and positive scalability by:
1. relaxing sequential specification, or
2. relaxing consistency condition, or
3. relaxing sequential specification and consistency condition.

Semantical Deviation
Semantical Deviation - Relaxed Sequential Specification

- Example FIFO queue:
- \( k \)-FIFO [PODC\, 11]: elements of a FIFO queue may be returned by a dequeue operation out of FIFO order up to a constant \( k \).
Semantical Deviation - Relaxed Consistency Condition

- Example linearizability:
- $k$-linearizability [SBG10]: an operation may be exchanged in order with any of its $k$ subsequent operations
- quasi-linearizability [OPODIS 2010, Afek et al.]
Semantical Deviation

linearizable k-FIFO queue

$\iff$

k-linearizable FIFO queue
Scal: Scalable Components

enqueue/dequeue

Backoff

Load Balancer

p FIFO Queues
Scal: FIFO queue

Load Balancer

[Diagram showing a FIFO queue with elements 1, 3, 5, 2, 4, 6 connected with arrows indicating the order and direction of elements进出 queue. Elements 7 and 8 are also shown, possibly indicating more elements or the queue's state at certain moments.]
Scal: FIFO queue

head
1 3 5
tail
2 4 6

Load Balancer
7
8
Scal: FIFO queue

head → 1 → 3 → 5 → tail

head → 2 → 4 → 6 → tail

Load Balancer

[7]

[8]
Scal: FIFO queue
Scal: FIFO queue

Load Balancer
Scal: FIFO queue

Load Balancer

1 → 3 → 5 → 8

2 → 4 → 6 → 7
Scal: FIFO queue

Load Balancer
Scal: Load Balancer

- Metric for the quality of the load balancer: Maximum imbalance of operations of a given type performed on the p FIFO queues
- Perfect load balancing
- Randomized load balancing
- Multiple choice
## Scal: Load Balancer Overhead

<table>
<thead>
<tr>
<th>Select function</th>
<th>Low contention (1 thread)</th>
<th>High contention (24 threads)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect</td>
<td>51 ns</td>
<td>3113 ns</td>
</tr>
<tr>
<td>Random</td>
<td>59 ns</td>
<td>64 ns</td>
</tr>
<tr>
<td>2-Random</td>
<td>108 ns</td>
<td>259 ns</td>
</tr>
</tbody>
</table>
Experiments

• Setup
  • Four 6-core 2.1GHz AMD Opteron processors (24 cores)
  • 48GB of memory
  • Linux 2.6.32
  • gcc 4.3.3 with -O3 optimizations
  • Benchmark threads were executed with real-time priorities
Experiments

• Data structure access pattern: alternating enqueue and dequeue operations
• Workload: high contention and low contention
• Scal: lock-free Michael-Scott FIFO queue using perfect, random, and 2-random load balancer
• Other:
  • baseline: lock-based FIFO queue, lock-free Michael-Scott FIFO queue,
  • quasi-linearizable: random dequeue queue (RDQ), segment queue (SQ)
Experiment
High Contention
Experiment
Low Contention
Experiment
Low Contention
Conclusion

• Relaxing concurrent object semantics may be the key enabler for performance and positive scalability of concurrent objects on multi-core systems

• Scal provides positive scalability and performance under high contention for FIFO queues

• 2-random load balancer provides best compromise between scalability and semantical deviation
Conclusion

• Rigorous systems engineering community should be prepared for more complex concurrent data structure semantics that provide performance and scalability under high contention

• Future work:
  • Other data structures
  • Software transactional memory
Questions?