

1 Relaxation for Efficient Asynchronous Queues

2 Samuel Baldwin 



3 Bucknell University, USA

4 Cole Hausman 

5 Bucknell University, USA

6 Mohamed Bakr 

7 Bucknell University, USA

8 Edward Talmage  

9 Bucknell University, USA

10 1 Introduction

11 1.1 Related Work

12 2 Model and Definitions

13 2.1 Queue Definitions

14 3 Asynchronous FIFO Queues

15 3.1 Description

16 3.2 Algorithm

17 3.3 Correctness

18 3.4 Complexity

19 4 Asynchronous Out-of-Order Queues

20 4.1 Description

21 4.2 Algorithm

22 4.3 Correctness

23 4.4 Complexity

24 5 Conclusion



© Samuel Baldwin, Cole Hausman, Mohamed Bakr, Edward Talmage;
licensed under Creative Commons License CC-BY 4.0
42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:4



Leibniz International Proceedings in Informatics
LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

■ **Algorithm 1** Code for each process p_i to implement a Queue with out-of-order k -relaxed *Dequeue*, where $k \geq n$ and $l = \lceil k/n \rceil$

```

1: function ENQUEUE( $val$ )
2:    $EnqCount = 0$ 
3:    $updateTS(v_i)$ 
4:    $enq\_timestamp = v_i$ 
5:   send ( $EnqReq, val, i, enq\_timestamp$ ) to all processes
6: end function
7: function RECEIVE( $EnqReq, val, j, enq\_timestamp$ ) from  $p_j$ 
8:    $updateTS(v_i, v_j)$ 
9:   if  $enq\_timestamp$  not in Pending_Enqueues then
10:     $Pending\_Enqueues.insertByTS(enq\_timestamp, val)$ 
11:   end if
12:   send ( $EnqAck, i$ ) to  $p_j$ 
13: end function
14: function RECEIVE( $EnqAck$  from  $p_j$ )
15:    $EnqCount++ = 1$ 
16:   if  $EnqCount == n$  then
17:     if  $localQueue.size < k$  then
18:       send ( $EnqConfirm, enq\_timestamp$ ) to all processes
19:     end if
20:   end if
21:   return  $EnqResponse$ 
22: end function
23: function RECEIVE( $EnqConfirm, enq\_timestamp$  from  $p_j$ )
24:    $localQueue.insertByTS(Pending\_Enqueues.getByTS(enq\_timestamp))$ 
25:   if  $clean == true$  and  $localQueue.size() \leq k$  then  $\triangleright$  localQueues agree by this point
26:     let  $procNum = (localQueue.size() - 1 \bmod n)$ 
27:      $localQueue.label(p_{procNum}, localQueue.tail)$   $\triangleright$  I may have mangled this line
28:   end if
29: end function

```

Algorithm 2 Continued, part 2

```

1: function DEQUEUE
2:    $v_i + = 1$ 
3:   let  $Deq_{ts} = v_i$ 
4:   if  $localQueue.peekByLabel(p_i) \neq \perp$  then           ▷ Check that I didn't change this
5:     let  $ret = localQueue.deqByLabel(p_i)$ 
6:     send ( $Deq_f, ret, Deq_{ts}$ ) to all processes
7:   else
8:     send ( $Deq_s, null, Deq_{ts}$ ) to all processes
9:   end if
10: end function
11: function RECEIVE( $deq_f, val, Deq_{ts}$ ) from  $p_j$ )
12:   if  $j \neq i$  then  $localQueue.remove(val)$ 
13:   end if
14: end function
15: function RECEIVE( $deq_s, val, Deq_{ts}$  from  $p_j$ )
16:    $UpdateTs(v_i, Deq_{ts})$ 
17:   if  $Deq_{ts}$  is not in  $PendingDequeues$  then
18:      $PendingDequeues.insertByTs(createList(Deq_{ts}, p_{invoker}))$            ▷ Check line
19:   end if
20:   let  $p_{invoker} = p_j$  ▷ This doesn't make sense to me? What are you doing on this line?
21:   if  $Deq_{ts} \neq 0$  and  $Deq_{ts} < v_i$  then
22:     send ( $Unsafe, Deq_{ts}, i, p_{invoker}$ ) to all processes
23:   else
24:     send ( $Safe, Deq_{ts}, i, p_{invoker}$ ) to all processes
25:   end if
26: end function

```

Algorithm 3 Continued, part 3

```

1: function RECEIVE(Safe/Unsafe,  $Deq_{ts}$ ,  $j$ ,  $p_{invoker}$ )
2:   if  $Deq_{ts}$  not in PendingDequeues then
3:     PendingDequeues.insertByTs(createList( $Deq_{ts}$ ,  $p_{invoker}$ ))
4:   end if
5:   for confirmationList in PendingDequeues do
6:     if confirmationList.ts ==  $Deq_{ts}$  then
7:       if Unsafe then
8:         response = 2
9:       else
10:        response = 1
11:      end if
12:      confirmationList.list[j] = response
13:    end if
14:    propagateEarlierResponses(PendingDequeues)
15:  end for
16:  for (index, confirmationList) in PendingDequeues do
17:    if not confirmationList.contains(0) and not confirmationList.handled then
18:      pos = 0
19:      for response in confirmationList.list do
20:        if response == 2 then
21:          pos++ = 1
22:        end if
23:      end for
24:      confirmationList.handled = True
25:      updateUnsafes(Lists, index)
26:      ret = localQueue.deqByIndex(pos)
27:      labelElements( $p_{invoker}$ )
28:      if  $i == p_{invoker}$  then
29:        return ret
30:      end if
31:    end if
32:  end for
33: end function

```

▷ Not sure I left the nesting right on these.