# The Baskets Queue
## Monsoon 2020, IIIT Hyderabad
## Project Presentation Report

Lakshmi Madhuri Yarava[2018101116]

---

**The Baskets Queue**

Moshe Hoffman , Ori Shalev , and Nir Shavit

# Introduction

Overall, the most popular dynamic-memory lock-free FIFO queue algorithm in the literature remains the MS-queue algorithm of Michael and Scott.The best known concurrent FIFO queue implementation is the lock-free queue of Michael and Scott .But it doesn't offer parallelism than that provided by allowing concurrent access to head and tail.

So ,here we discuss "Baskets Queue" , a new, highly concurrent lock-free linearizable dynamic memory FIFO queue. It allows parallelism in design of Concurrent shared queues.In Enqueue operation,it basically creates baskets of mixed-order items instead of the standard totally ordered list.In our new "basket" approach, instead of the traditional ordered list of nodes, the queue consists of an ordered list of groups of nodes (baskets). The order of nodes in each basket need not be specified, and in fact, it is easiest to maintain them in LIFO order.

So we can do many enqueue and dequeue operations parallelly Operations in different baskets can be executed parallelly.

## 1.1 Baskets Queue

Linearizability was introduced as a correctness condition for concurrent objects.For a FIFO queue, an execution history is linearizable if we can pick a point within each enqueue or dequeue operation's execution interval so that the sequential history defined by these points maintains the FIFO order.

---

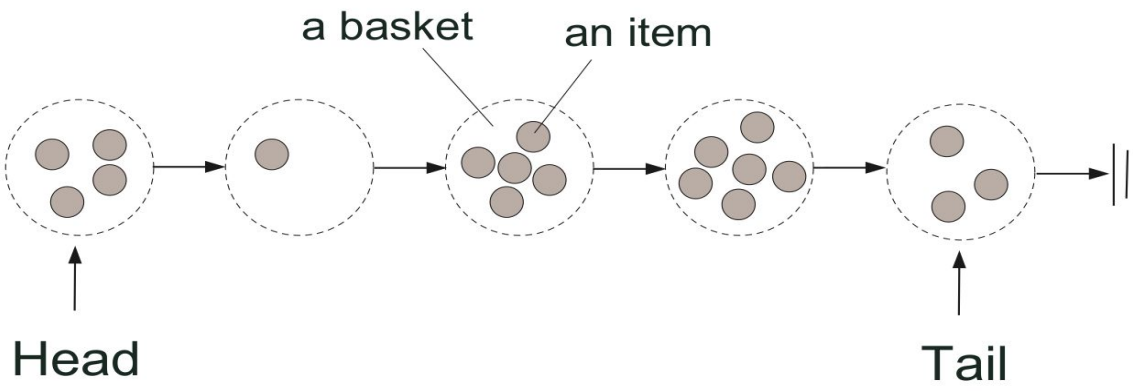
For a basket having one or more nodes,baskets must fulfill these basic rules:-

- 1.Each basket has a time interval in which all its nodes' enqueue operations overlap.
- 2.The baskets are ordered by the order of their respective time intervals.
- 3.For each basket, its nodes' dequeue operations occur after its time interval.
- 4.The dequeue operations are performed according to the order of baskets.

Two properties define the FIFO order of nodes:

- 1. The order of nodes in a basket is not specified.
- 2. The order of nodes in different baskets is the FIFO-order of their respective baskets.

The basic idea behind these rules is that setting the linearization points of enqueue operations that share an interval according to the order of their respective dequeues, yields a linearizable FIFO-queue.




We need not specify the order between nodes in Basket . So,nodes in the same basket can be dequeued in any order,as order of enqueue operations can be "fixed" to meet the dequeue operations order.

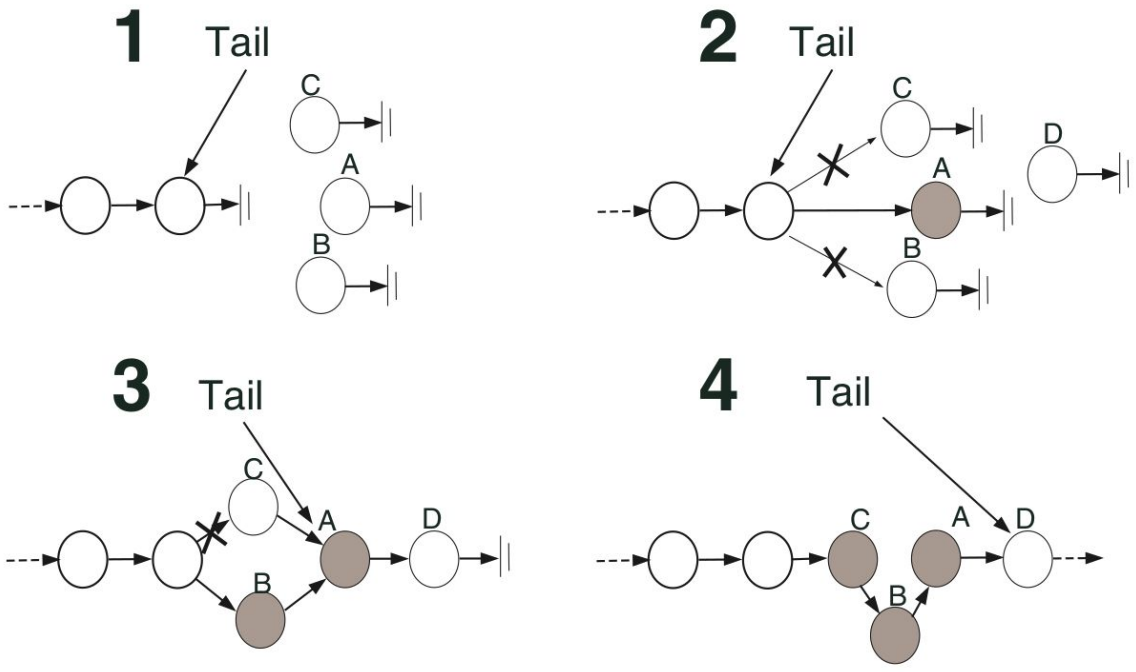
“Compare and Swap Operation” is an atomic structure used in Multithreading. It compares the contents of a Memory Location with a given value and “if they are same” , it modifies the contents of that memory location to a given new value.

Detecting Enqueue operations overlap:-

In Algorithms such as MS-queue or Optimistic,threads enqueue items by applying ”CAS Operation” overlap in time.Hence, all the threads that fail to CAS on the tail-node of the queue may be inserted into the same basket.The time usually spent on backing-off before trying to link onto the new tail, can now be utilized to insert the failed operations into the basket, allowing enqueues to complete sooner. In the meantime, the next successful CAS operations by enqueues allow new baskets to be formed down the list, and these can be filled concurrently

## 1.2 Enqueue and Dequeue Functions

To enqueue a node , a thread first tries to link the new node to the last node.If it failed to do so, then another thread has already succeeded  by inserting another one.



Explanation:-

1)Here , we have three threads A,B,C . Each of them checks if tail's next node is null so that they can insert a new node and tries to atomically change it to point to its new node's address

2)Out of them,Thread A succeeded to  enqueue it's node whereas B,C threads are failed and hence they will try to get themselves inserted

3)Thread B succeeded to enqueue, meanwhile,at the same time thread D calls the enqueue operation, and finishes successfully to enqueue onto the tail.

4)Then Thread C got enqueued successfully.

To dequeue a node, a thread first reads the head of the queue to obtain the oldest basket. It may then dequeue any node in the oldest basket.

## 1.3 Code

Since we employ CAS operations in our algorithm, ABA issue will definitely be there and we use tag mechanism to avoid that.

*struct pointer_t {*

*<ptr, deleted, tag>: <node_t *, boolean, unsigned integer>*

*};*

We define a node with value and pointer for next

*struct node_t {*

*data_type value;*

*pointer_t next;*

*};*

Also,we define a Queue with head and tail.

*struct queue_t {*

*pointer_t tail*

*pointer_t head*

*};*

Enqueue Operation:-

To enqueue a node,

- ➔ first the thread reads current tail (E04)
- ➔ If Tail is the last node then it tries to atomically link the new node to the last node (E09).
- ➔ We basically do CAS operation to compare and swap a new value in queue.So here,we try to do CAS operation with Tails next pointer and a new node (E09)
- ➔ If the CAS operation succeeded the node was enqueued successfully, and the thread tries to point the queue's tail to the new node (E10), and then returns
- ➔ It re-reads the next pointer that points to the first node in the basket, and as long as no node in the basket has been deleted (E13)
- ➔ Thus, the thread tries to insert the new node to the basket (E12-E18)
- ➔ If the tail did not point to the last node, the last node is searched (E20-E21), and the queue's tail is fixed.

E01: nd = new_node()

E02: nd->value = val

E03: repeat:

E04:tail = Q->tail

E05:next = tail.ptr->next

E06:if (tail == Q->tail)):

E07:if (next.ptr == NULL):

E08:nd->next = <NULL, 0, tail.tag+2>

E09:if CAS(&tail.ptr->next, next, <nd, 0, tail.tag+1>):

E10:CAS(&Q->tail, tail, <nd, 0, tail.tag+1>

E11:return True

E12:next = tail.ptr->next

E13:while((next.tag==tail.tag+1) and (not next.deleted)):

E14:backoff_scheme()

E15:nd->next = next

E16:if CAS(&tail.ptr->next, next, <nd, 0, tail.tag+1>):

E17:return True

E18:next = tail.ptr->next;

E19:else:

E20:while ((next.ptr->next.ptr != NULL) and (Q->tail==tail)):

E21:next = next.ptr->next;

E22:CAS(&Q->tail, tail, <next.ptr, 0, tail.tag+1>)

Enqueue Operation:-

To dequeue a node,

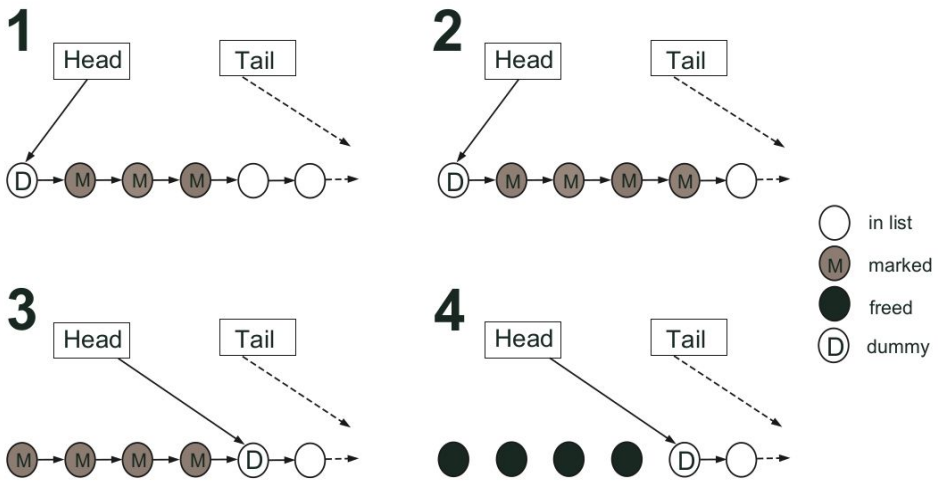
- ➔ a thread reads the current state of the queue (D01-D04) like head and tail
- ➔ Then it checks for its consistency (D05)
- ➔ D06-D08 , it checks if there is any possibility of queue being empty by comparing head's next and tail
- ➔ If the head and tail of the list point to the same node (D06), then either the list is empty (D07) or the tail lags.
- ➔ If not,last node is searched (D09-D10) and the tail is updated(D11)
- ➔ If the head and the tail point to different nodes, then the algorithm searches for the first unmarked node between the head and the tail (D15-D18).
- ➔ If a non-deleted node is found, its value is first read (D24) before trying to logically delete it (D25)
- ➔ If our deletion is successfully completed then dequeue is done
- ➔ Before returning, if the deleted node is far enough from the head (D26), the free chain method is performed (D27). If while searching for a non-deleted node the thread reached the tail (D21) the queue's head is updated (D22)

1)Here,three nodes are logically deleted

2)the first non-deleted node is deleted

3) the head is advanced and frees those logically deleted nodes by moving forward

4)the chain of deleted nodes can be reclaimed




➔

Different Benchmarks and Results:-

Comparisons between MS Queue,Optimistic Queue,Baskets Queue:--

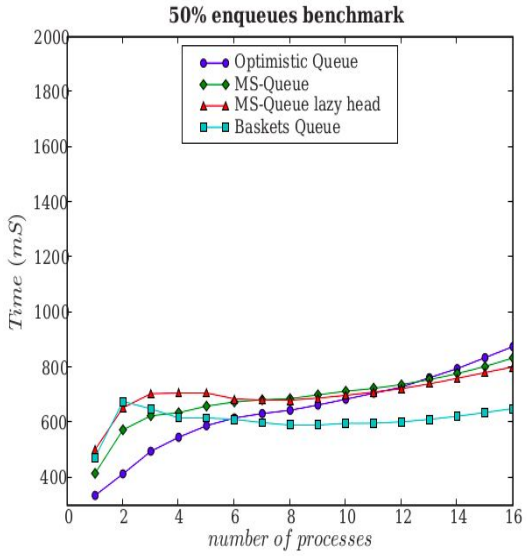



**Fig. 9.** The 50 % enqueues benchmark

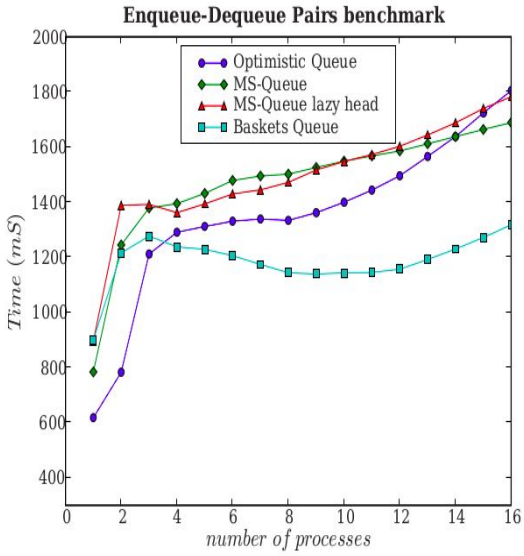



**Fig. 10.** The Enqueue-Dequeue pairs benchmark

From above figures,It can be seen that high levels of concurrency have only moderate effects on the performance of the Baskets Queue. The Baskets Queue is up to 25% faster than the other algorithms. This can be explained by the load on the tail of all the data-structures but the baskets queue, whereas in the baskets queue the contention on the tail is distributed among several baskets. However, at lower concurrency levels, the optimistic approach is superior because the basket-mechanism is triggered upon contention.

Below is a typical snapshot of the Baskets Queue on the 50% enqueues benchmarks. The basket sizes vary from only 1 to 3 nodes. In the average case, an enqueue operation will succeed to enqueue after at most 3 failed CAS operations. The baskets sizes are smaller than 8 nodes as one would expect them to be, because the elements are inserted into the baskets one by one. This unnecessary synchronization on the nodes of the same basket imposes a delay on the last nodes to be inserted.

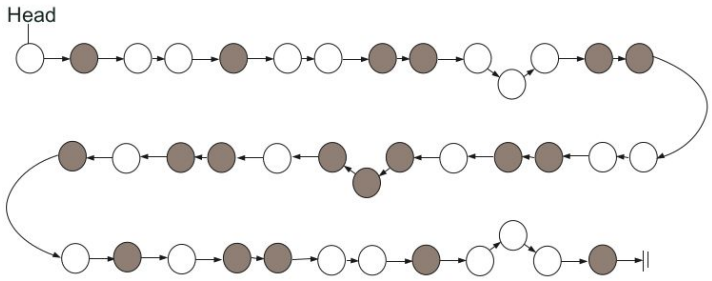



**Fig. 15.** A typical snapshot of the queue (16 processes)

I have explained the implementation part of code in another PDF

## References:-

https://people.csail.mit.edu/shanir/publications/Baskets%20Queue.pdf