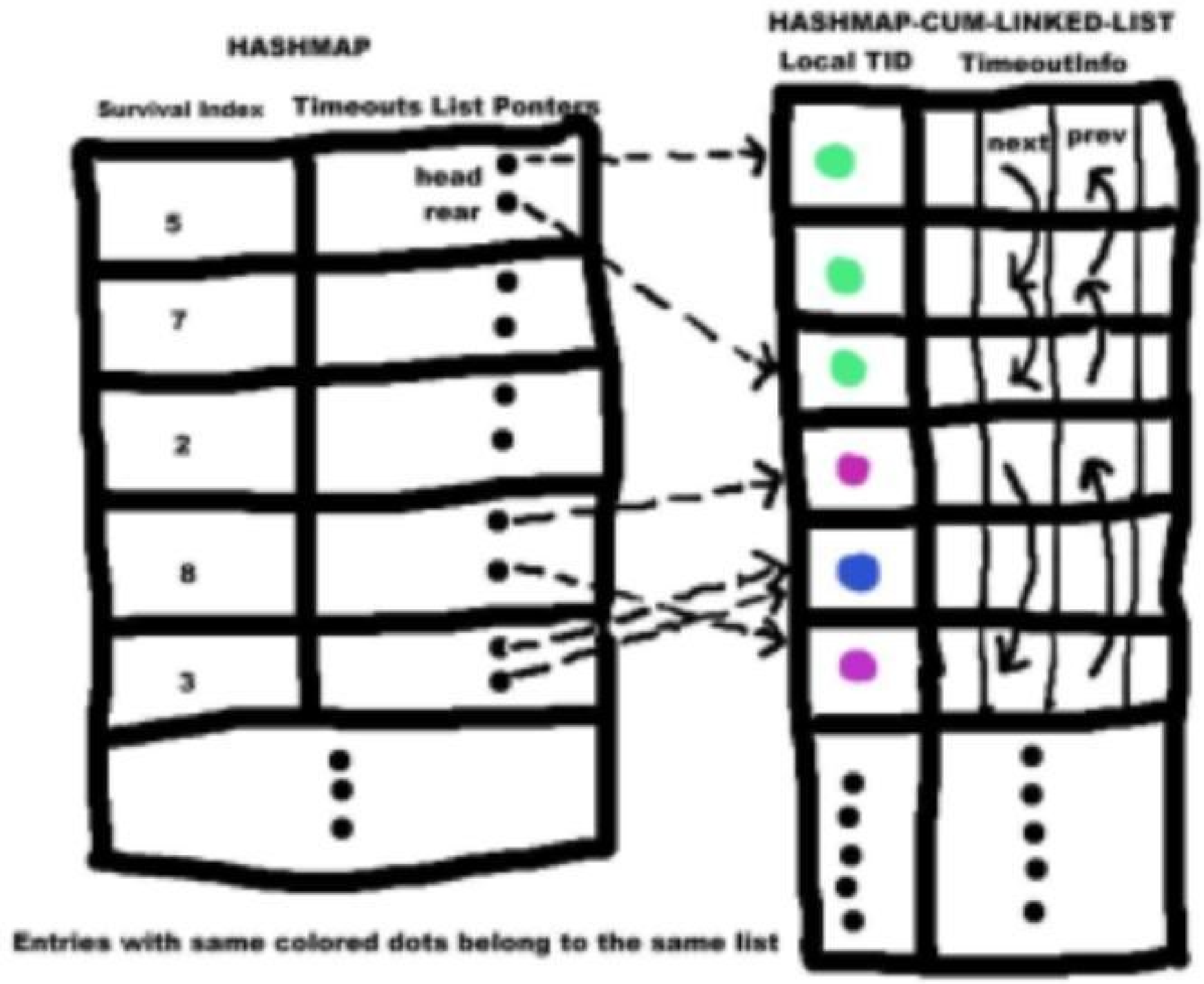
TIMEOUT MANAGER REDESIGN PROPOSAL



setTimeout( )

2a.Maximum number of invocations of timeout thread that a transaction will survive is calculated as below:
Survival Index
(or)
maximum number of invocations of TimeoutThread this txn may survive =
CEIL{(timeout_for_this_txn - time_remaining_for_the_next_invocation_of_timeout_thread) / TIMEOUT_INTERVAL} + 1

An illustration:

TO interval

|<---------->|<----------->|<----------->|

----> <----|

left right

remainder remainder

|***********************************|

max transaction duration(or) timeout

|-------|

(time remaining for the next invocation of TO thread)

|-----------------|

(timeout for this txn - time remaining for the next invocation of timeout thread)

| | (i.e only 2 integers are in the range)

CEIL{(timeout for this txn - time remaining for the next invocation of timeout thread)/TIMEOUT_INTERVAL}

| | | (i.e add one more to compensate for left remainder)

CEIL{(timeout for this txn - time remaining for the next invocation of timeout thread)/TIMEOUT_INTERVAL} + 1

```
---------addTimeoutInfo( )--------------------------------------------
2b.synchronized(pendingTimeouts)
{
    If value obtained in 2(a) exists within HashMap of SurvivalIndices
    {
        the transaction info will be put into localTID->TimeoutInfo Map and appended to alread existing linked list for that value
        update head and rear pointers for this list
    }
    else
    {
        put a new survival index entry in SurvivalIndex HashMap;put timeout info into localTID->TimeoutInfo Map and add the same as the first element of the timeout info list
        initialise head and rear pointers to this list
    }
}
-------addTimeoutInfo( )----------------------------------------------
```

TimeoutManager.run( )

```
while(true)
{
    Sleep for TIMEOUT_INTERVAL

    For all survival Indices in pendingTimeouts
    {
        increment timeoutThreadInvocationCount for this entry by 1
        if(survivalIndex == timeoutThreadInvocationCount) /* there is a TIMEOUT */
        {
```

```
synchronized(TimeoutListPointer for this list) // as it is the wrapping object of this list
{
All the transactions in the TimeoutInfo linked list corresponding to this survival index have to be rolled
back/recovered accordingly, if still alive. The head and rear pointers to the list are maintained by
TimeoutsListPtr object which is part of the survival index entry.
Remove the survival index entry and accompanying timeout info list
}
}

}
}
```

SURVIVAL INDEX equals( ) method

a.Consider the following scenario of SurvivalIndex Map contents. Figure in parantheses is timeout thread invocation count. Letters in RHS of arrow signify transactions corresponding to this survival index:

1) Before invocation of Timeout thread
5(0) =>a b c
4(0) =>d e
6(0) =>f g h i

2) After invocation of timeout thread
5(1) => a b c
4(1) => d e
6(1) => f g h i
Now say a transaction j with SurvivalIndex 4 has to be added to this map. But to which list? It is decided by the equals( ) method of SurvivalIndex object. Two survival indices A and B are equal iff,
A.survivalIndex - A.timeoutThreadInvocationCount == B.survivalIndex - B.timeoutThreadInvocationCount
Using above criterion, transaction j with SurvivalIndex 4 (and timeoutThreadInvocationCount 0, as it is nascent) is equal to SurvivalIndex 5 (since 5 - 1 = 4 (survivalIndex - timeoutThreadInvocationCount))
So our map becomes,
5(1) => a b c j
4(1) => d e
6(1) => f g h i

SURVIVAL INDEX hashcode( ) method

A new static variable globalTimeoutThreadInvocationCount is introduced in Timeout Manager for the purpose described below. It will be incremented on each invocation of Timeout Thread. It is publicly accessible.
Survival Index should also implement hashCode( ) method since survivalIndex varies after each Timeout Thread invocation. As a result, hashCode( ) must be returned in such a way as to get the correct entry. Consider the following :

Logical view of survivalIndex -> TimeoutsListPtr map

SurvivalIndex ------ TimeoutsListPtr
1 ....
11 ....
21 ....
4 ....

14 ....
24 ....
7 ....
17 ....
27 ....
37 ....

Internal representation of the above logical view (Before Timeout Thread Invocation)

Let array_length be 10.
globalTimeoutThreadInvocationCount == 0

array_index(hashCode % array_length) all survival indices with same value of (hashCode % array_length)
1 --------------------------- 1 -> 11 -> 21
4 --------------------------- 4 -> 14 -> 24
7 --------------------------- 7 -> 17 -> 27 -> 37

Internal representation of the above logical view (After Timeout Thread Invocation)

globalTimeoutThreadInvocationCount == 1

array_index(hashCode % array_length) all survival indices with same value of (hashCode % array_length)
1 --------------------------- 0 -> 10 -> 20
4 --------------------------- 3 -> 13 -> 23
7 --------------------------- 6 -> 16 -> 26 -> 36

Now, suppose a transaction with survival index 13 has to be added to localTID->TimeoutInfo Map, we have to get a hashCode in such a way that we end up in index 4 (instead of 3). So hashCode must return the value of the expression:
this.survivalIndex - this.timeoutThreadInvocationCount + TimeoutManager.globalTimeoutThreadInvocationCount
For our example, above expression becomes 13 - 0(because it has just been created) + 1 = 14
So, (hashCode( ) % array_length) becomes, 14 % 10 = 4 which is the required index.
Then the equals method starts a search down the list for this index, and decides that there is an entry 13 already using logic described in previous section on equals( ) method.

BENEFITS

a. timeout is calculated **a priori** instead of checking at every timeout thread invocation.

b. overhead to check each and every element of the pendingTimeouts for timeout is minimized to large extent by iterating only through survival indices on each timeout thread invocation. Since in a typical stress scenario, large number of transactions will be created in each timeout interval, they all would have same survival index (provided same timeout is set for all txns which will mostly be the case) and thus would be part of a single timeout info list. Thus **length of survival index list is directly proportional to survival index instead of total number of transactions.**

EXPERIMENT

A performance comparison of existing timeout manager and the implementation of above design :

Number of requests = 30 users * 30 iterations

Average Time for Existing Timeout manager - 27 seconds
Average Time for new implementation - 24 seconds

NOTE: The above values are the best among many runs for the corresponding implementation