Lawn - an Unbound Low Latancy Timer Data Structure for Large Scale Systems

Adam Lev-Libfeld

Tamar Labs
Tel-Aviv, Israel
adam@tamarlabs.com

December 4, 2017

Abstract

As demand for Real-Time applications rise among the general public, the importance of enabling large scale, unbound algorithms to solve conventional problems with low to no latency is critical for product viability Timers algorithms are prevalent in the core mechanisms behind of operating systems, network protocol implementation, stream processing and several data base capabilities This paper presents an algorithm for a low latency, unbound range timer structure, based upon the well excepted Timing Wheel algorithm. Using a set of queues hashed by TTL, the algorithm allows for simpler implementation, minimal overhead and no degradation in performance in comparison to current state of the algorithms under typical use cases.

Index Terms **- Stream Processing, Timer Wheel, Dehydrator, Callout facilities, protocol implementations, Timers, Timer Facilities.**

1 Introduction

2 Model

In similar manner to previous work[1], the model discussed in this paper shall consist of the following components, each corresponding with a different stage in the life cycle of a timer in the data store:

$$\begin{tikzcd}
 & StartTimer \arrow[d] & \\
 & PerTickBookkeeping \arrow[dl] \arrow[dr] \arrow[loop right] & \\
DeleteTimer & & TimerExpired
\end{tikzcd}$$

StartTimer(TTL,timerId{,Payload}): This routine is called by the client to start a timer that will expire in after the TTL has passed. The client is also expected to supply a *timer ID* in order to distinguish it from other timers in the data store. Some implementations also allow the client to provide a *Payload*, usually some form of a callback action to be performed or data to be returned on timer expiration.

PerTickBookkeeping: This routing encompasses all the actions, operation and callbacks to be performed as part of timer management and expiration check every interval as determined by the data store granularity. Upon discovery of an outstanding timer to expire *TimerExpired* will be initiated by this routine.

DeleteTimer(timerId): The client may call this utility routine in order to remove from the data store an outstanding timer (corresponding with a given *timer ID*), this is done by calling *TimerExpired* for the requested timer before *PerTickBookkeeping* had marked it to be expired.

TimerExpired(timerId): Internally invoked by either *PerTickBookkeeping* or *DeleteTimer* this routine entails all actions and operations needed in order to remove all traces of the timer corresponding with a given *timer ID* from the data store and invoking the any callbacks that were provided as *Payload* during the *StartTimer* routine.

Since payload and callback behavior varies significantly between different data store implementations, the store of such data can be achieved for $O(1)$ using a simple hash map, and the handling of such callbacks is highly (or even embarrassingly) parallel this paper will disregard this aspect of timer stores.

3 Timer Store Implementations

4 The Lawn Data Structure

4.1 Intended Use and

5 Comparison and Reflection

6 An Algorithmic View

7 Conclusion

References

[1] George Varghese and Anthony Lauck. Hashed and hierarchical timing wheels: efficient data structures for implementing a timer facility. *IEEE/ACM transactions on networking*, 5(6):824–834, December 1997.