Experiment

A small experiment involving a bunch of threads incrementing a shared counter one per iteration is used to benchmark the response time for different locks. The experiment is run on the c220g2 in cloudlab, which has two Intel E5-2660 v3 10-core CPUs at 2.60 GHz (Haswell EP). Figure 1 is the ECDF of the response time for each lock when running with 40 threads.

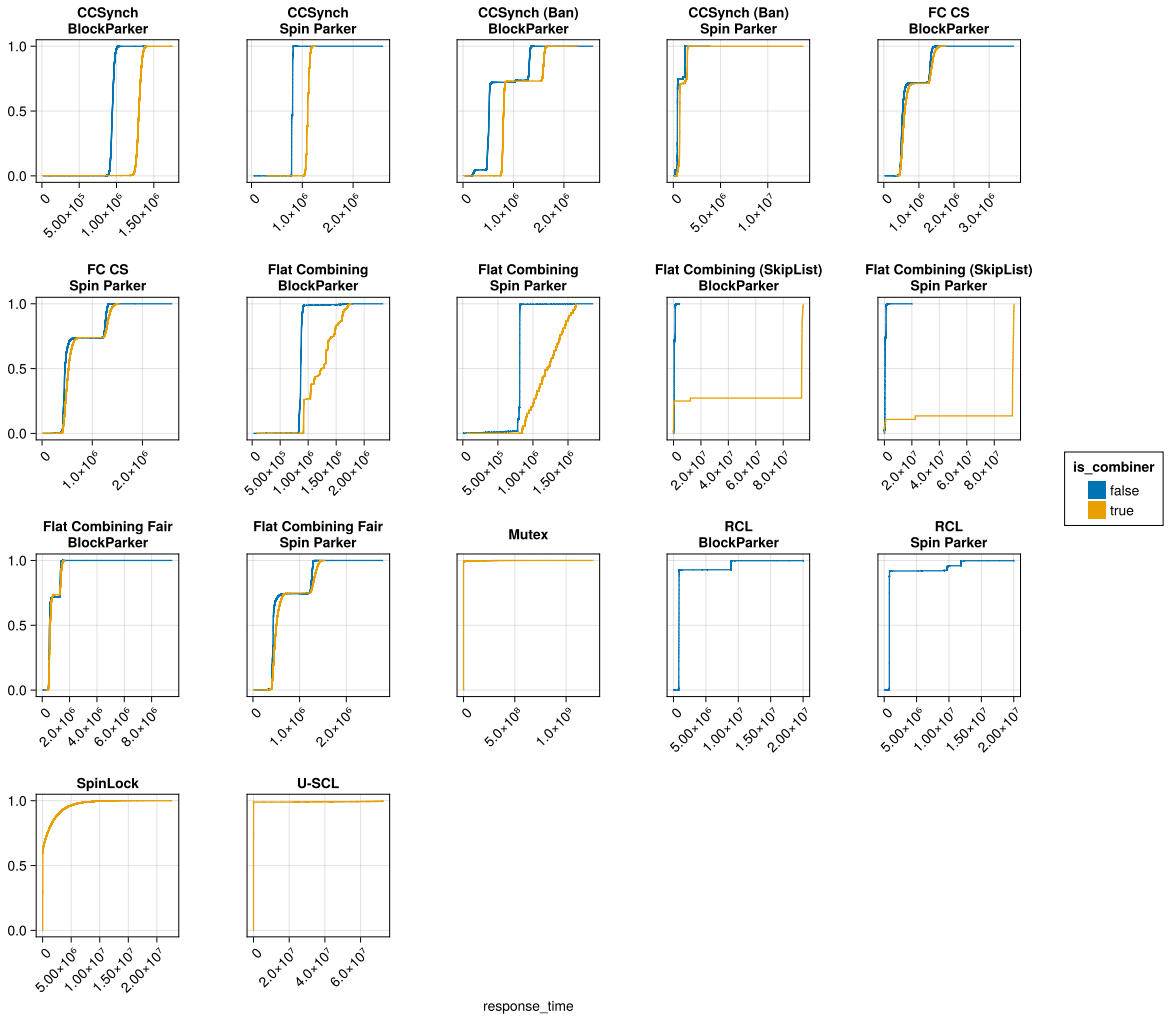

Figure 1: Response Time ECDF for 40 threads

The *x-axis* is not linked across plot to allow seeing the individual patten of the ECDF for each lock. The duration of each critical section is super short, which also creates some unique patterns for the experiment.

Mutex/SpinLock/U-SCL

Mutex has the a very large range on the *x-axis*. Thus it is possible for a thread to wait very long even though all they wish to do is to provide a small increment to the shared counter.

SpinLock has a interesting pattern of the response time. The curve of the ECDF is very smooth, which is quite interesting. Other than that it has a similar behavior as Mutex.

U-SCL's response time behavior has been well analysis in its paper. The response time is very small for most scenerio given the presence of lock slice but it can be very large when waiting for a lock slice.

Delegation Locks

The main focus of this write up is about the discussion of response time of delegation locks, and specifically discussing the difference of response time when a thread is and is not combining. The yellow ecdf curve is the response time of delegation lock when a thread is combining, and the blue ecdf curve is the response time of delegation lock when a thread is not combining.

We can see the response of delegation locks are not skewed to the left as **Mutex/SpinLock/U-SCL**. This is likely because threads are not able to execute their critical section right after entering the locks (given that most delegation locks provides expected acquisition fairness). Though U-SCL also provides similar guarantee, the presence of lock slice changes it behavior. If we have a ticket lock, I would anticipate a similar behavior with it (**TODO**).

Flat-Combining

In the center of the graph, we can see the behavior of various variants of Flat Combining lock. Specifically, we see a pattern of *linear* climbing when a thread is a combiner for the original Flat-Combining. This might indicate some *probability distribution* of the number of threads that are waiting, since combiner of Flat-Combining traverse all threads node and try to execute their critical section. The blue line also extend to the right most as the yellow line, as the combiner may skip some thread when they are preparing to the job, which means they have to wait the next pass (or the thread lies at the end of the *linked list*).

The behavior of Flat-Combining (Banning) (**Flat Combining Fair** in the graph at the left bottom) has a similar distribution regardless whether a thread is combining, which is kindly interesting. Note that the *x-axis* scale of blocking version is different from the one of spinning version (and also different from Flat-Combining). Theoredically, I anticipate to see similar result as Flat-Combining, but since the critical section is too short, the calculation of usage might get errored, causing some thread got banned unexpectedly.

The *kink* is interesting, but I cannot think of a reasonable explanation for it. This *kink* also appears for CC-Synch (Banning), so I believe might be caused by the huristic algorithm of banning or that the experiment is running in a 2-core system, which has *NUMA* behavior. Though since other locks doesn't have that behavior I would doubt the huristic algorithm is more like the cause.

CC-Synch

On the top of the graph, we can see that CC-Synch is providing a more concentrated distribution of response time, both combining or not. The panalty of combining is smaller than expected, but probably due to the fact that the critical section is too short.

The *kink* is also presented in CC-Synch (Banning), and we can see that some small number of combiner will have very long waiting time. This is because the job limit is setted based on volunteering time as Flat-Combining (Skip-List), but surprisingly much less combiner is combining to the limit, which differs significantly from Flat-Combining (Skip-List). This might be due to the banning policy, which reduces the number of waiters.

Flat-Combining (Skip-List)

At the rightest of the second row, we can see the behavior of Flat-Combining (Skip-List) shows some expected pattern. The non-combining threads are having a seemingly short response time, and the combining threads are having a relatively large waiting time. The promise of Flat-Combining (Skip-List) is to provide a **CFS**-like scheduling policy, with a bound of volunteering time as combiner. Even though the combiner may experience long waiting time, it is bounded by the maximum servicing time of the lock.

Future Work

The experiment is pretty rough, and is conducted for a very short critical sections. The following is a list of future work that I would like to do.

1. One that has variable length of critical section, and see how the response time changes. Maybe this time more than just 1 vs 3 ratio.
2. One that has slightly longer critical section with similar critical section length.