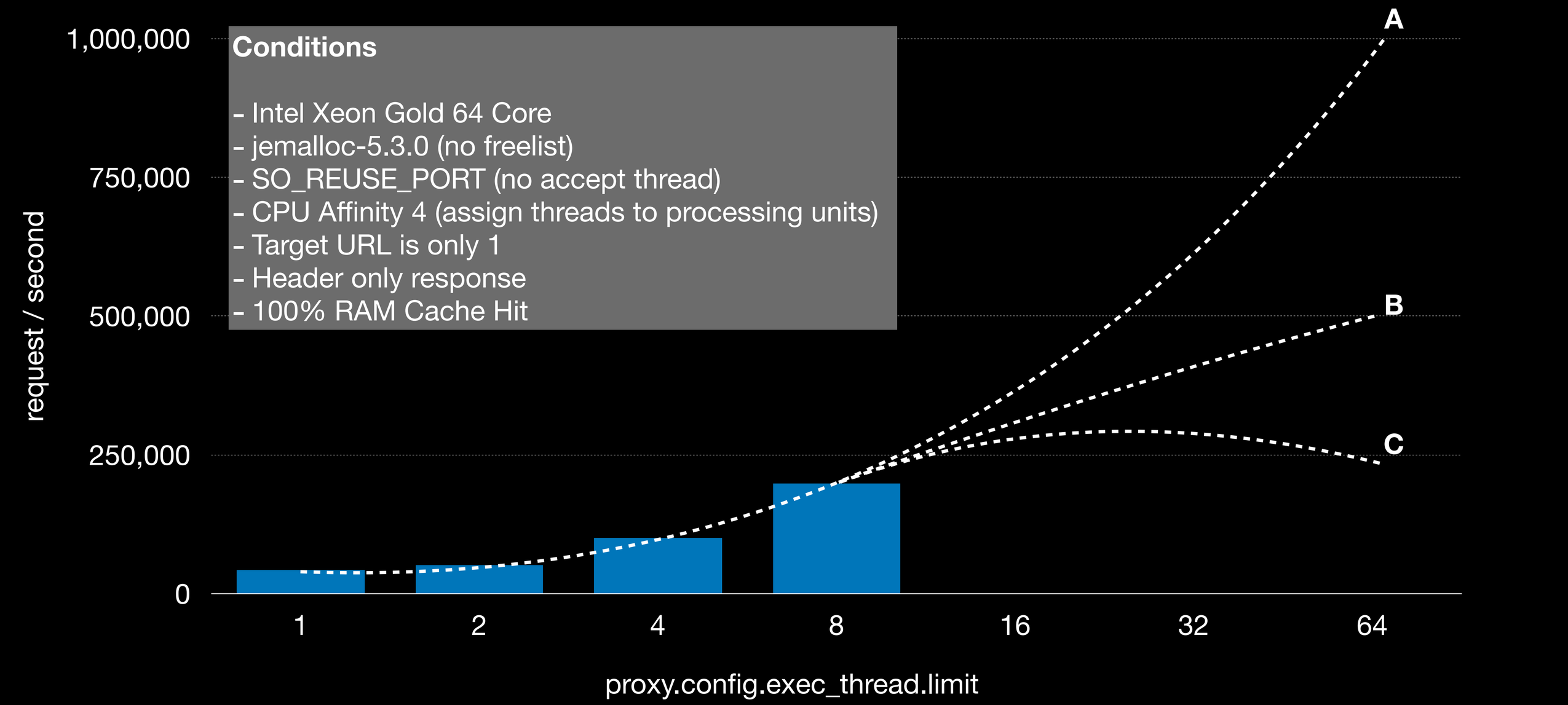
Accelerating Traffic Server

ATS Spring 2023 Summit

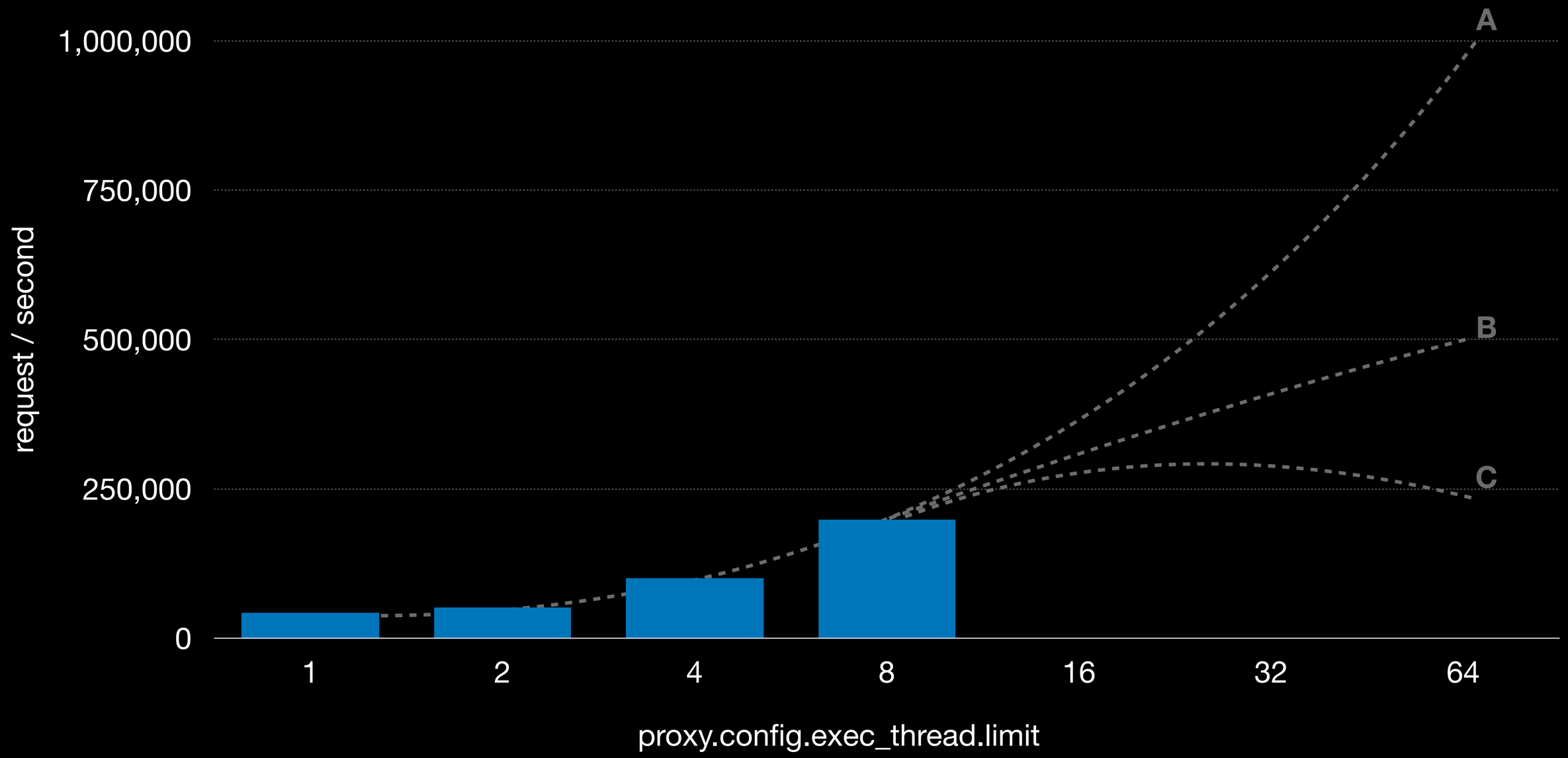
Masaori Koshiba (masaori@apache.org)

Quiz

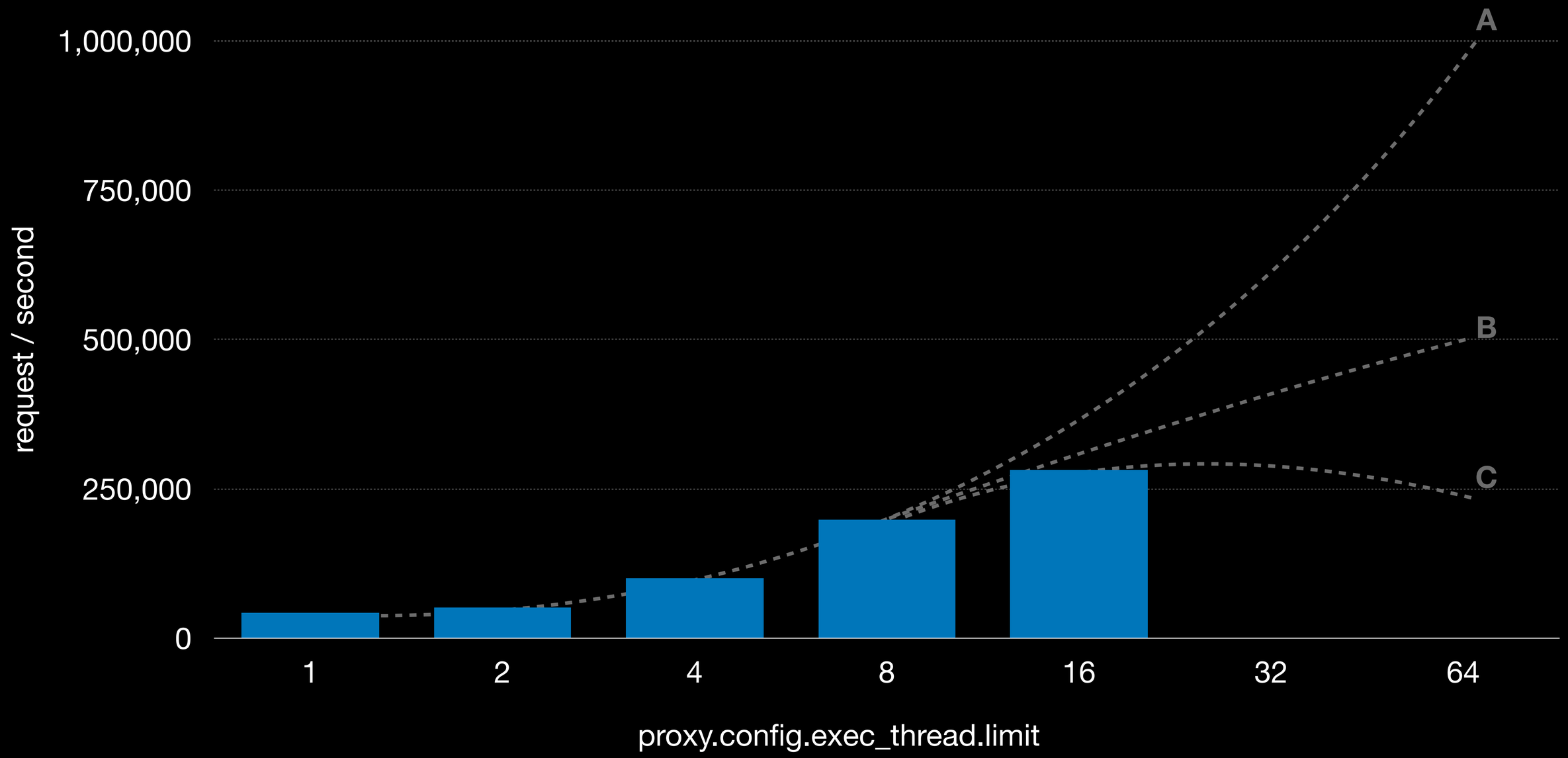
Q: Does ATS scale with the number of cores?



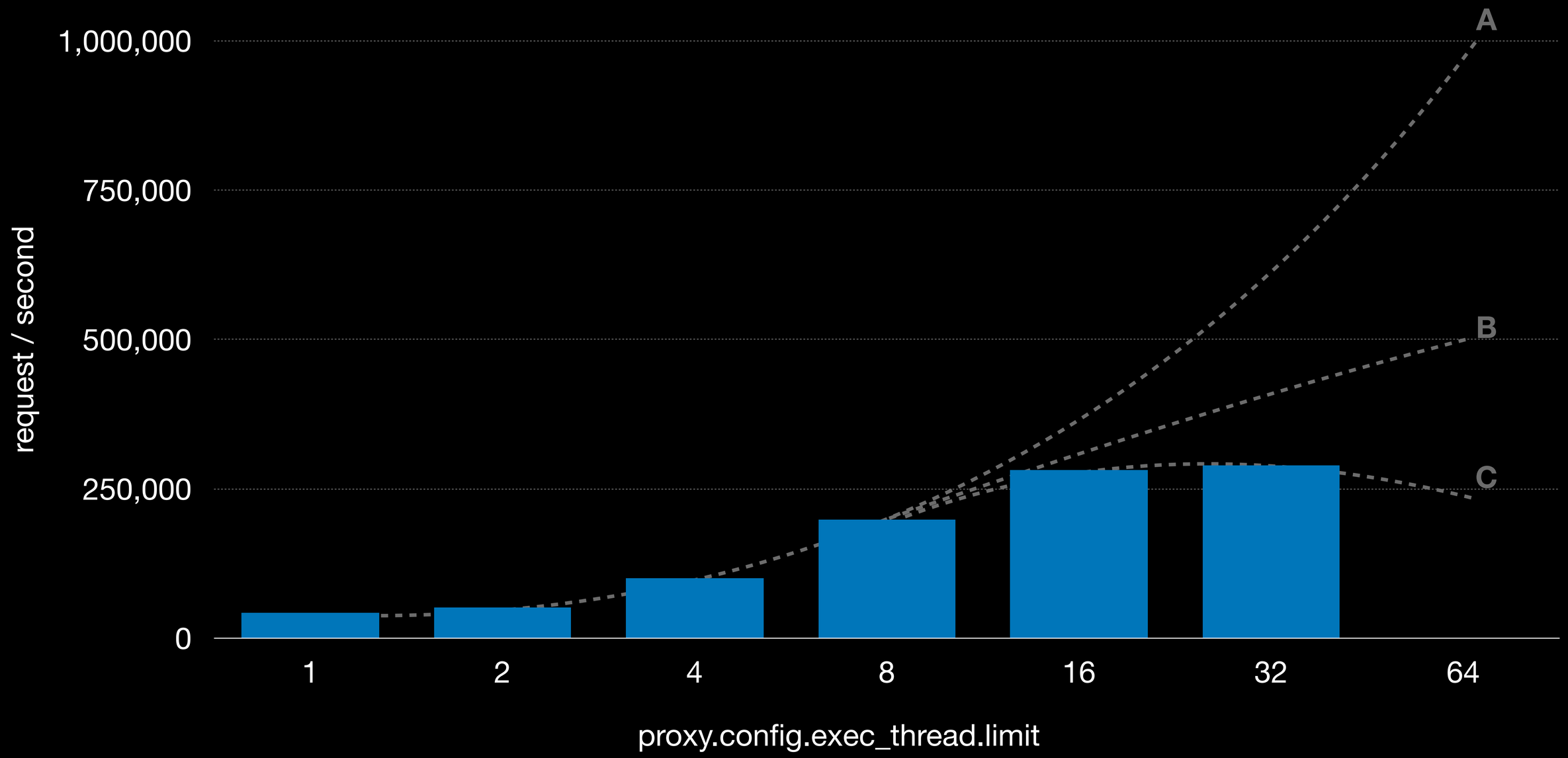
Q: Does ATS scale with the number of cores?



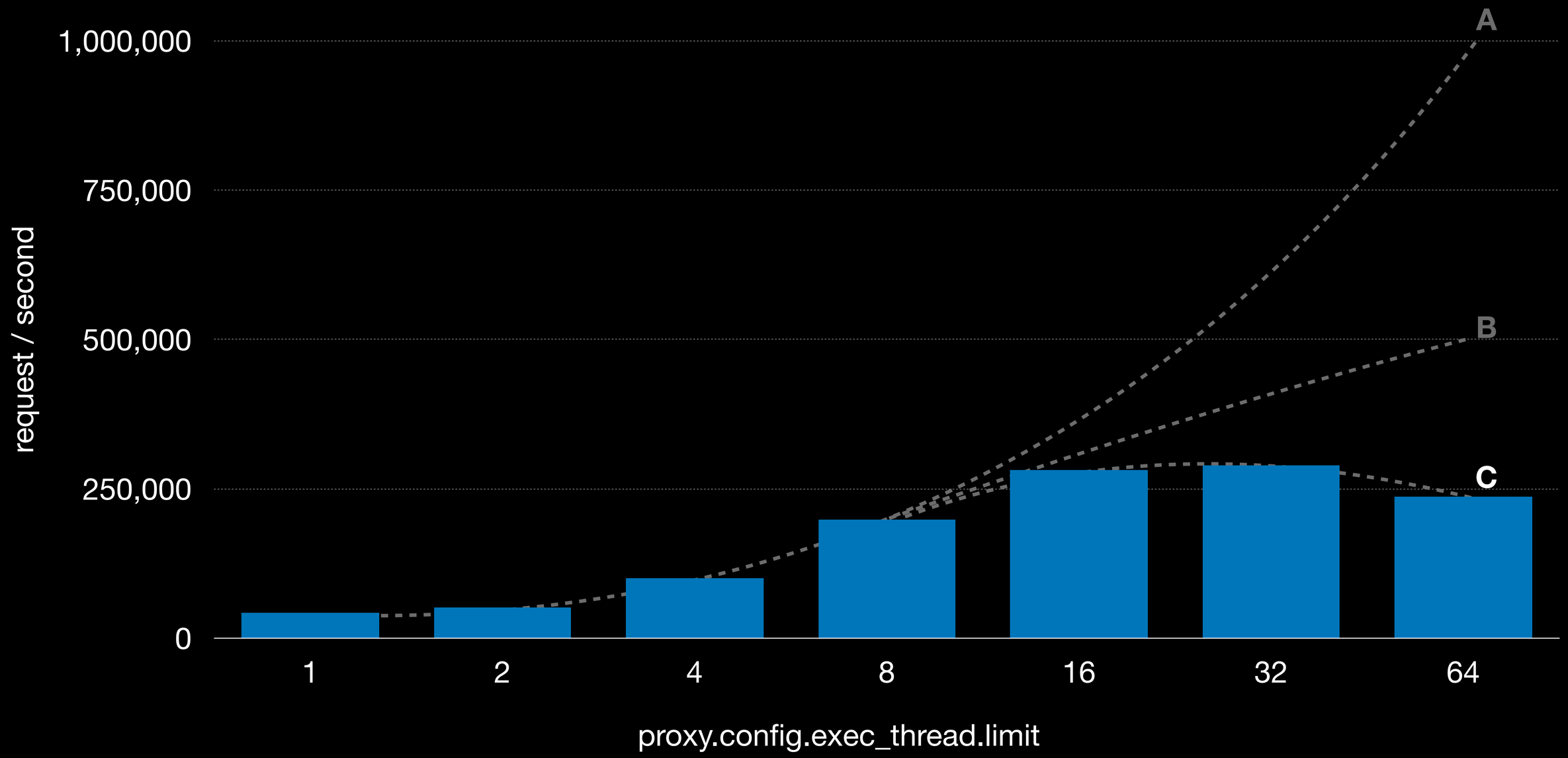
Q: Does ATS scale with the number of cores?



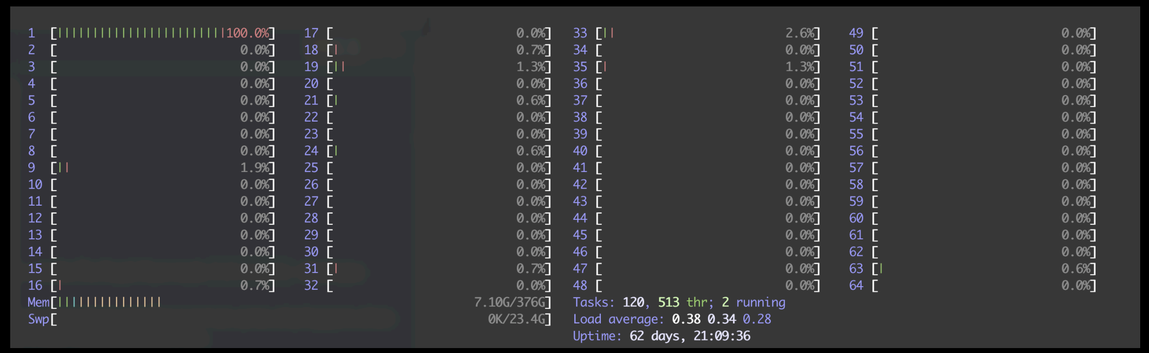
Q: Does ATS scale with the number of cores?



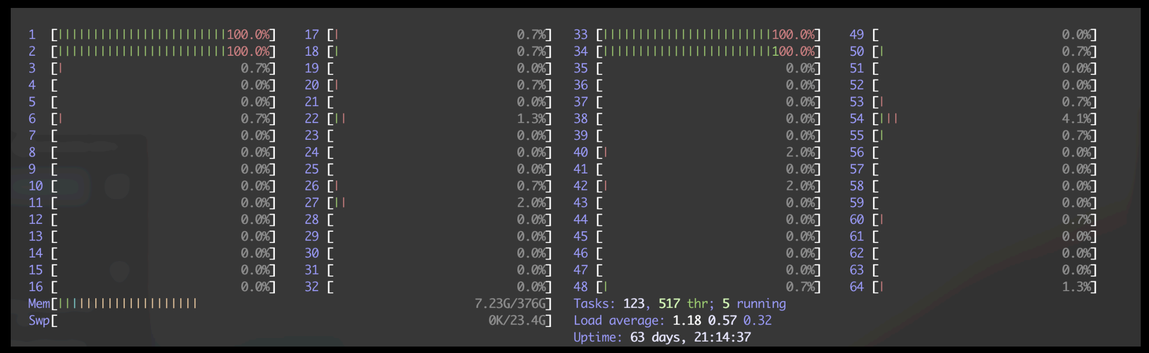
Q: Does ATS scale with the number of cores?



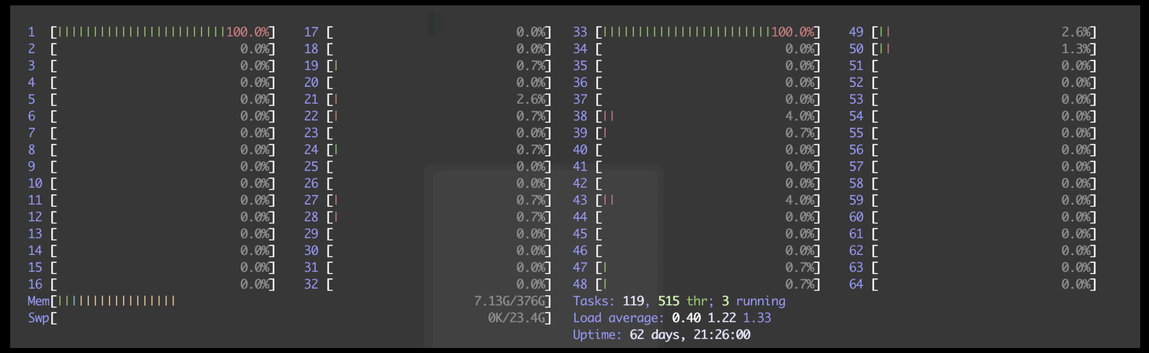
1 thread



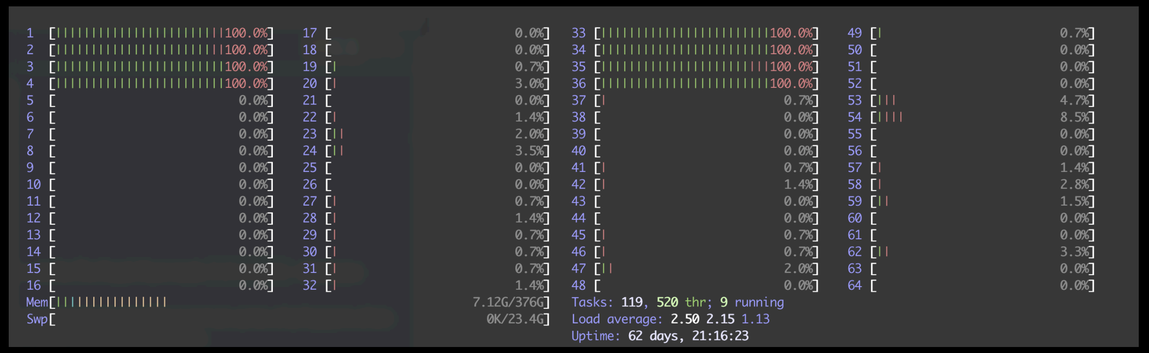
4 threads



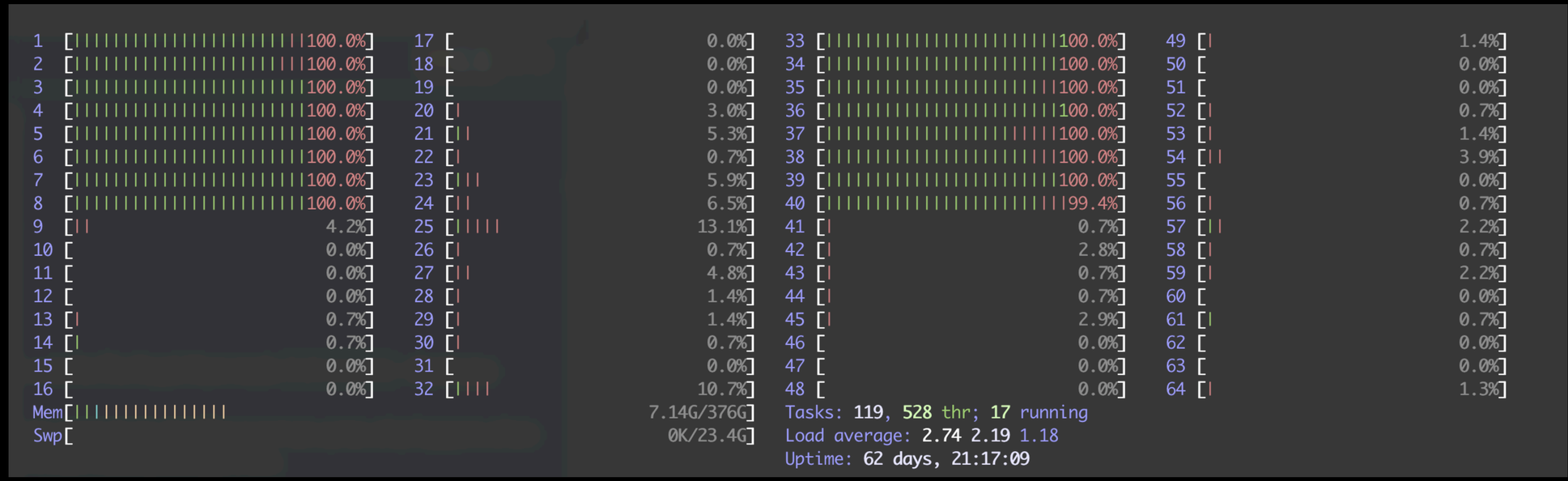
2 threads



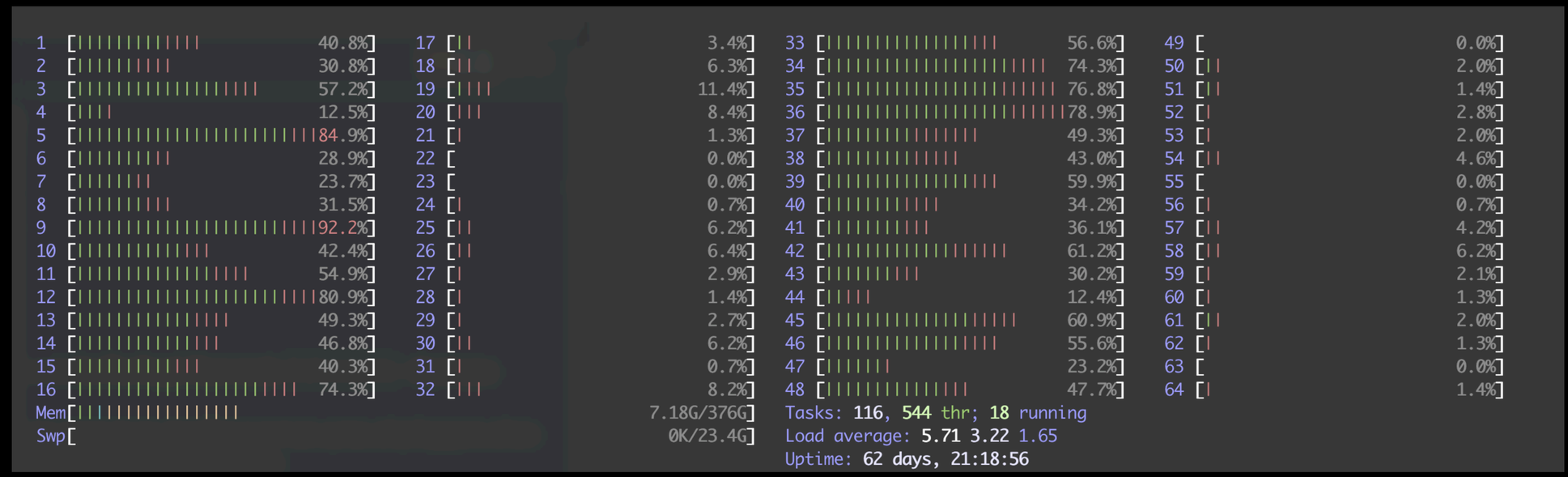
8 threads



16 threads

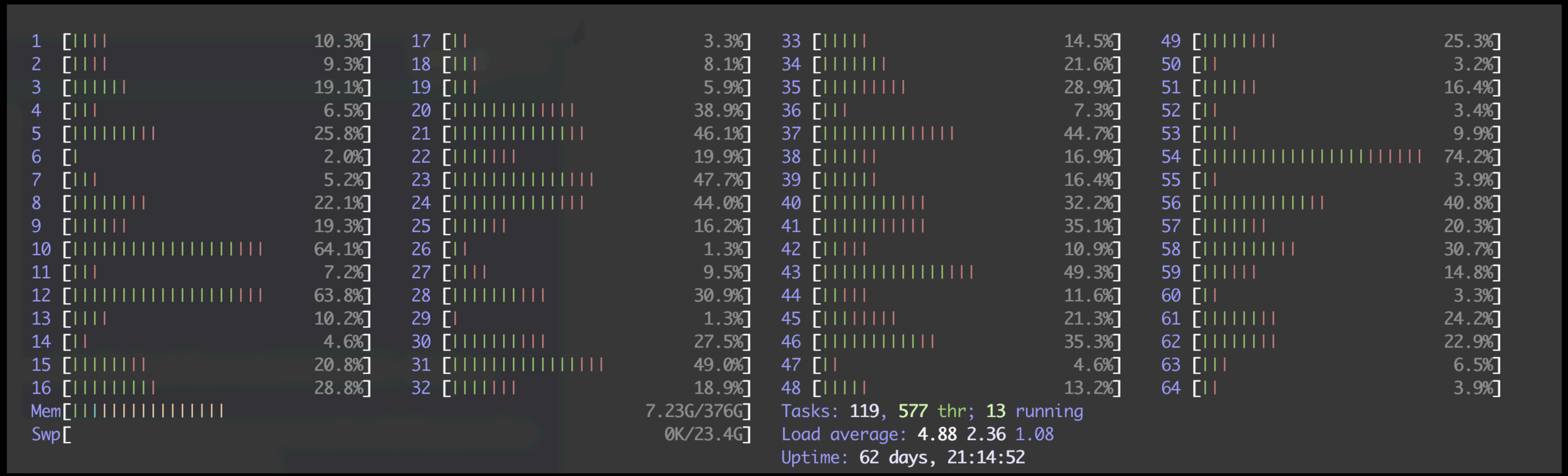


32 threads



"Underutilized" Issue

64 threads



Profiling ATS

Cache Enabled (almost 100% Hit)

```
thread 38875
        mutex [unknown] ::: wait time 0.00us ::: hold time 24692.07us ::: enter count 3051 ::: try-lock failure count 13693
                CacheVC::openReadStartHead(int, Event*)+0xd8 [traffic_server] (55e891616368)
                EThread::process_event(Event*, int)+0x276 [traffic_server] (55e89170b586)
                EThread::execute_regular()+0x33d [traffic_server] (55e89170bf0d)
                EThread::execute()+0x171 [traffic_server] (55e89170c361)
                spawn_thread_internal(void*)+0x55 [traffic_server] (55e89170a8a5)
                start_thread+0xc5 [libpthread-2.17.so] (7f1216cf8ea5)

        mutex [unknown] ::: wait time 0.00us ::: hold time 10263.61us ::: enter count 1372 ::: try-lock failure count 6294
                CacheVC::openReadClose(int, Event*)+0xab [traffic_server] (55e89161a1db)
                EThread::process_event(Event*, int)+0x276 [traffic_server] (55e89170b586)
                EThread::execute_regular()+0x33d [traffic_server] (55e89170bf0d)
                EThread::execute()+0x171 [traffic_server] (55e89170c361)
                spawn_thread_internal(void*)+0x55 [traffic_server] (55e89170a8a5)
                start_thread+0xc5 [libpthread-2.17.so] (7f1216cf8ea5)

        mutex [unknown] ::: wait time 0.00us ::: hold time 3041.71us ::: enter count 337 ::: try-lock failure count 1391
                Cache::open_read(Continuation*, ats::CryptoHash const*, HTTPHdr*, OverridableHttpConfigParams const*, CacheFra
                CacheProcessor::open_read(Continuation*, HttpCacheKey const*, HTTPHdr*, OverridableHttpConfigParams const*, lo
                HttpCacheSM::open_read(HttpCacheKey const*, URL*, HTTPHdr*, OverridableHttpConfigParams const*, long)+0x9d [tr
                HttpSM::do_cache_lookup_and_read()+0x169 [traffic_server] (55e89147d929)
                HttpSM::set_next_state()+0x4d1 [traffic_server] (55e8914820f1)
                HttpSM::set_next_state()+0xb48 [traffic_server] (55e891482768)
                HttpSM::set_next_state()+0xa06 [traffic_server] (55e891482626)
```

Profiling ATS

Cache Enabled (almost 100% Hit)

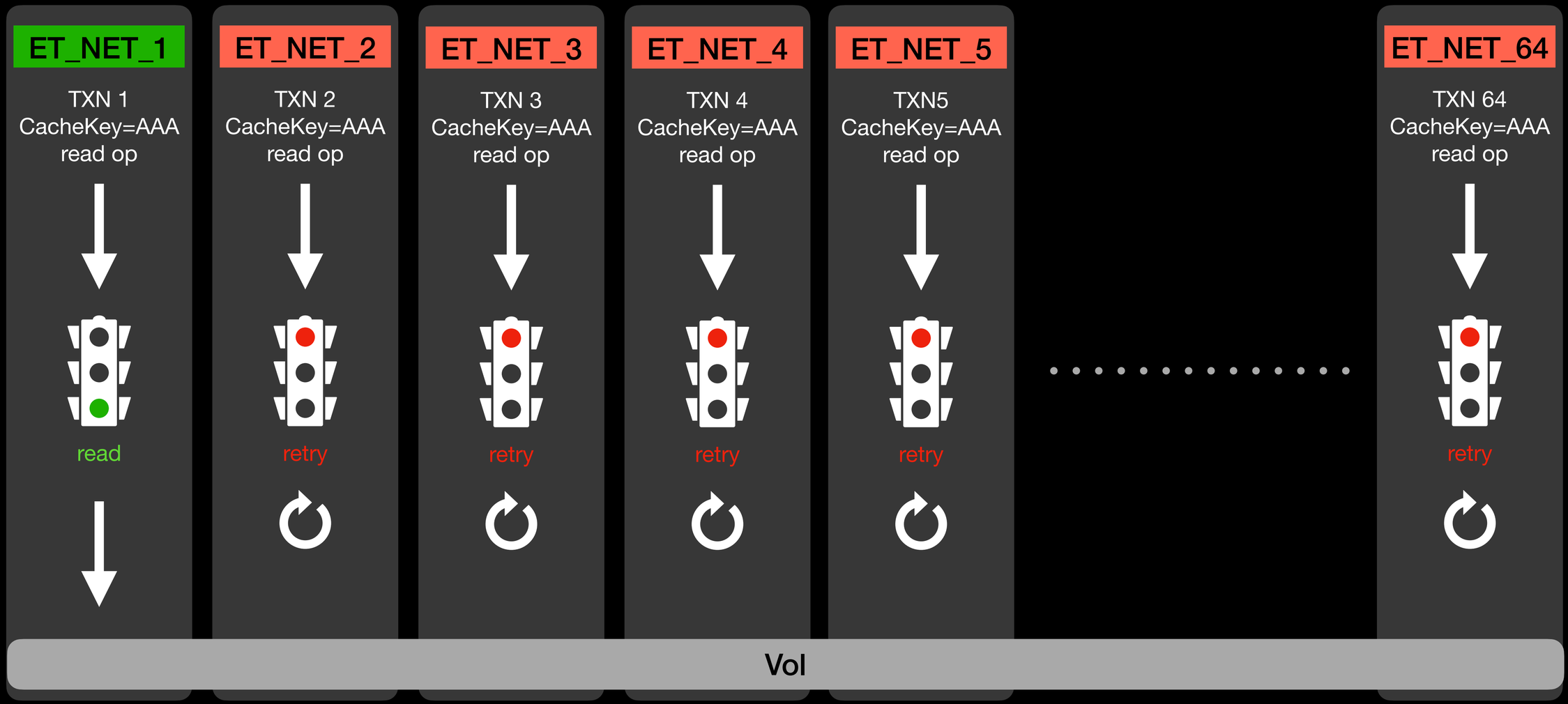
```
thread 38875
        mutex [unknown] ::: wait time 0.00us ::: hold time 24692.07us ::: enter count 3051 ::: try-lock failure count 13693
                CacheVC::openReadStartHead(int, Event*)+0xd8 [traffic_server] (55e891616368)
                EThread::process_event(Event*, int)+0x276 [traffic_server] (55e89170b586)
                EThread::execute_regular()+0x33d [traffic_server] (55e89170bf0d)
                EThread::execute()+0x171 [traffic_server] (55e89170c361)
                spawn_thread_internal(void*)+0x55 [traffic_server] (55e89170a8a5)
                start_thread+0xc5 [libpthread-2.17.so] (7f1216cf8ea5)
```

```
int
CacheVC::openReadStartHead(int event, Event *e)
{
  ...
  {
    CACHE_TRY_LOCK(lock, vol->mutex, mutex->thread_holding);
    if (!lock.is_locked()) {
      VC_SCHED_LOCK_RETRY();
    }
```

https://github.com/apache/trafficserver/blob/c983006eccbce9365224c2cd30372528ac8df843/iocore/cache/CacheRead.cc#L1069

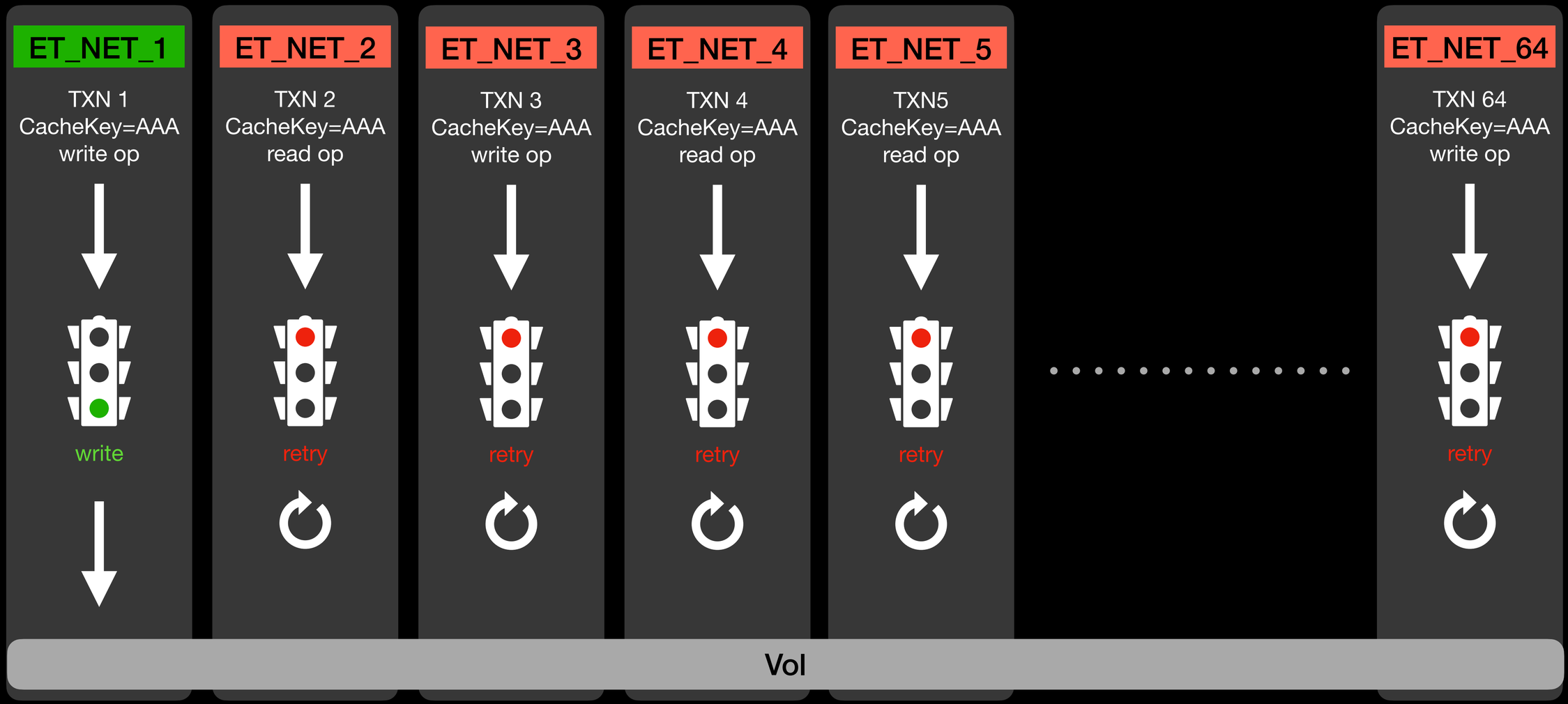
```
ailure count 6294

lure count 1391
rams const*, CacheFra
nfigParams const*, lo
onst*, long)+0x9d [tr

                HttpSM::set_next_state()+0xb48 [traffic_server] (55e891482768)
                HttpSM::set_next_state()+0xa06 [traffic_server] (55e891482626)
```

Try Lock Contention of Vol Mutex

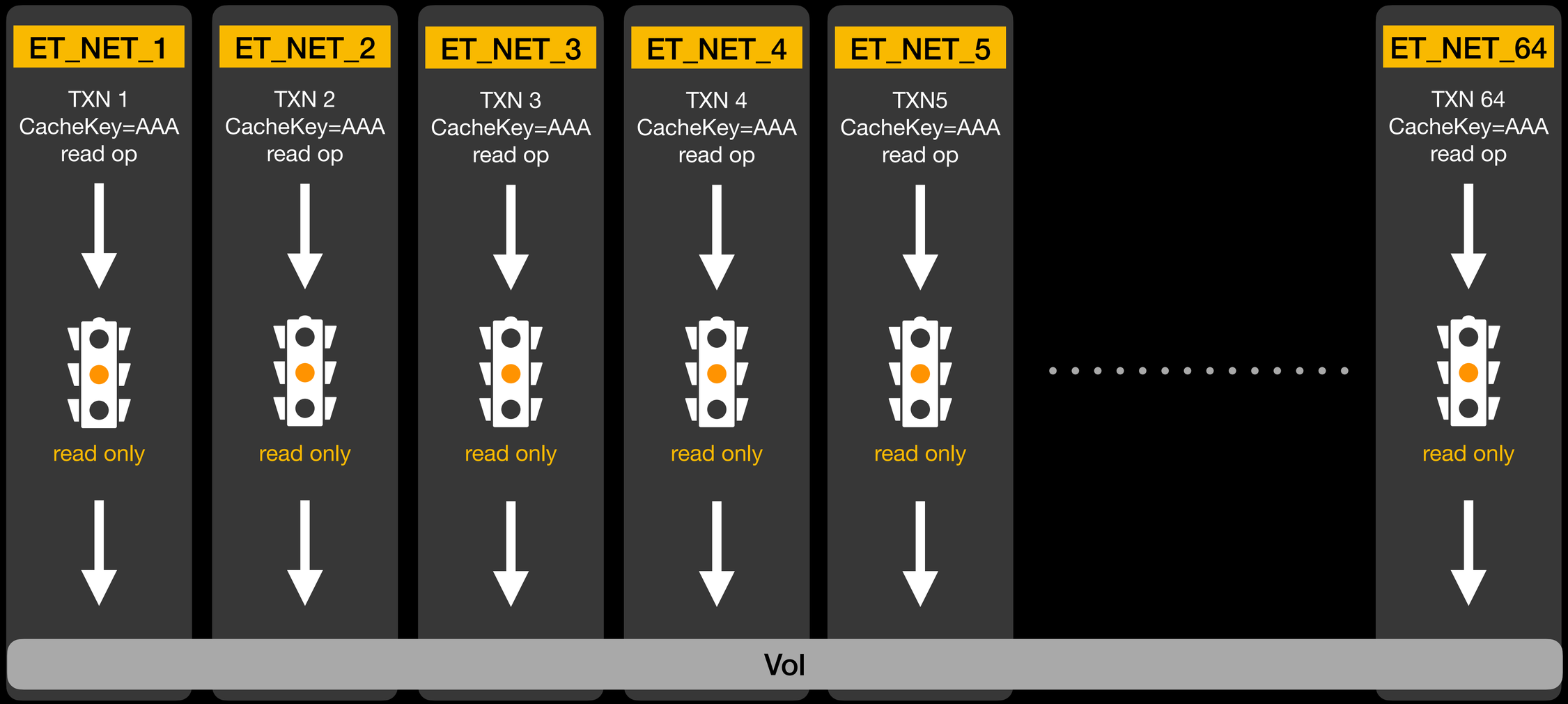


Reader-Writer Lock?

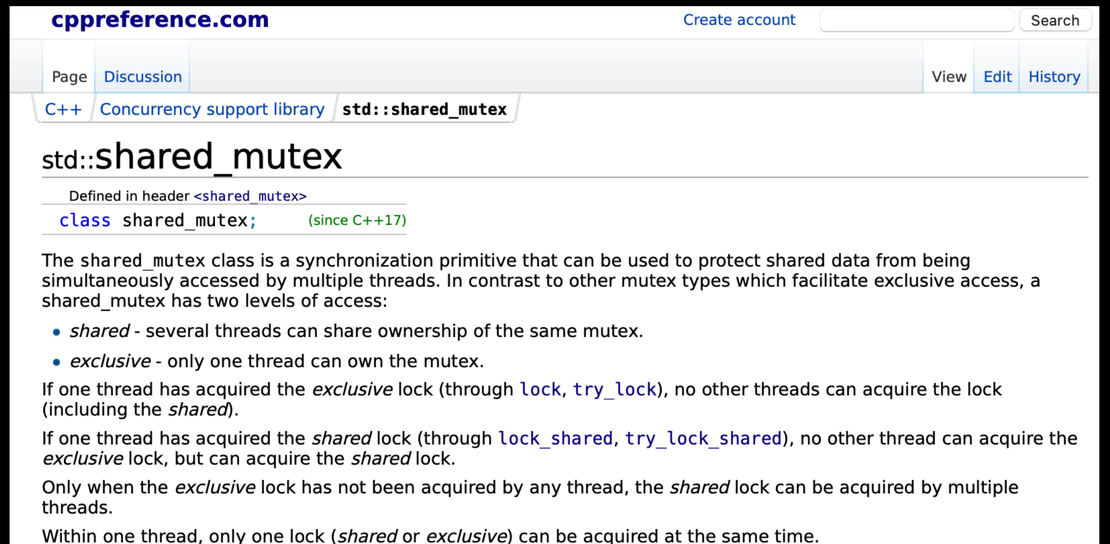
Reader-Writer Lock (Cache Write)



Reader-Writer Lock (Cache Read)



TrafficServer (9.2.0)
+ Replace Vol Mutex with std::shared_mutex



cppreference.com

Create account

Search

Page | Discussion | View | Edit | History

C++ | Concurrency support library | std::shared_mutex

std::shared_mutex

Defined in header <shared_mutex>

```
class shared_mutex;      (since C++17)
```

The shared_mutex class is a synchronization primitive that can be used to protect shared data from being simultaneously accessed by multiple threads. In contrast to other mutex types which facilitate exclusive access, a shared_mutex has two levels of access:

- *shared* - several threads can share ownership of the same mutex.
- *exclusive* - only one thread can own the mutex.

If one thread has acquired the *exclusive* lock (through lock, try_lock), no other threads can acquire the lock (including the *shared*).

If one thread has acquired the *shared* lock (through lock_shared, try_lock_shared), no other thread can acquire the *exclusive* lock, but can acquire the *shared* lock.

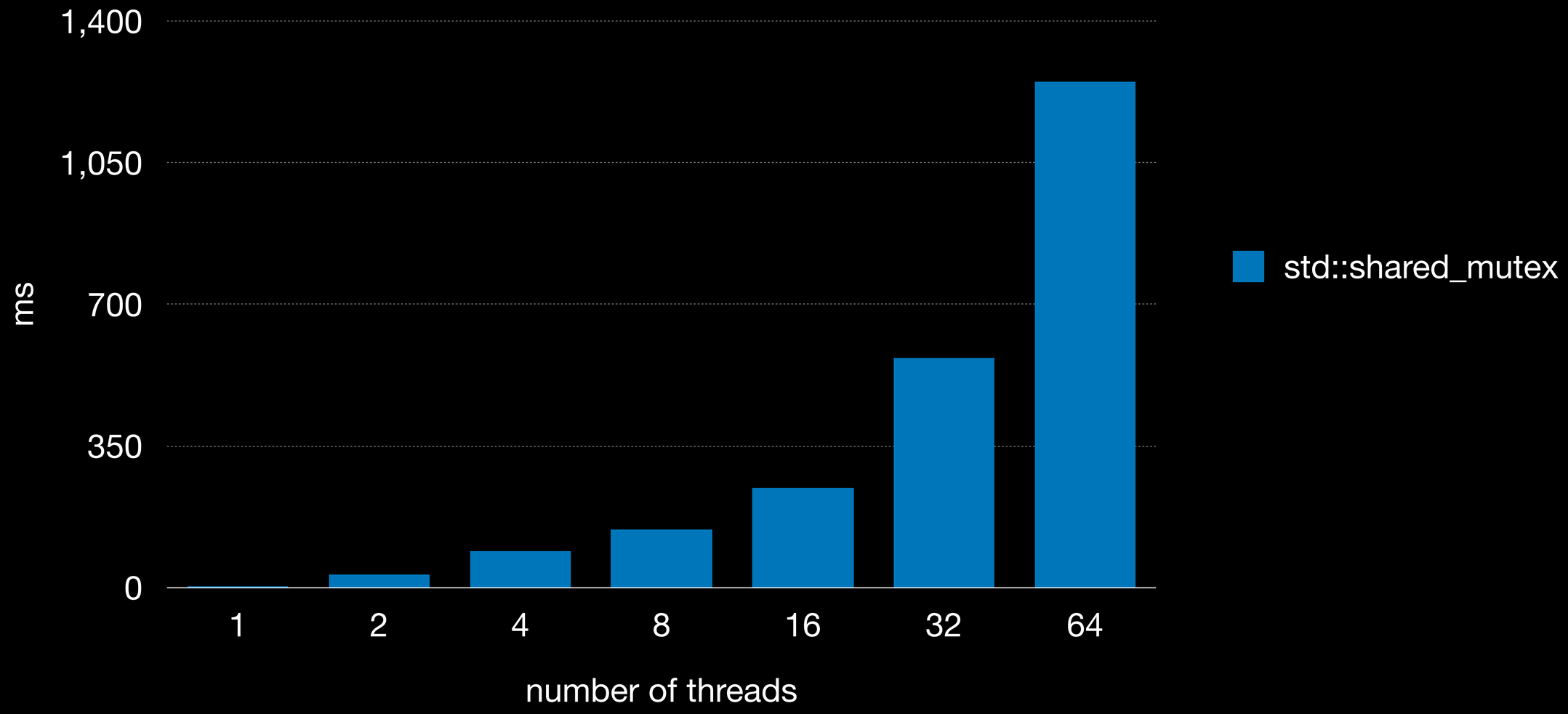
Only when the *exclusive* lock has not been acquired by any thread, the *shared* lock can be acquired by multiple threads.

Within one thread, only one lock (*shared* or *exclusive*) can be acquired at the same time.

Micro-Benchmark of std::shared_mutex

read-read performance

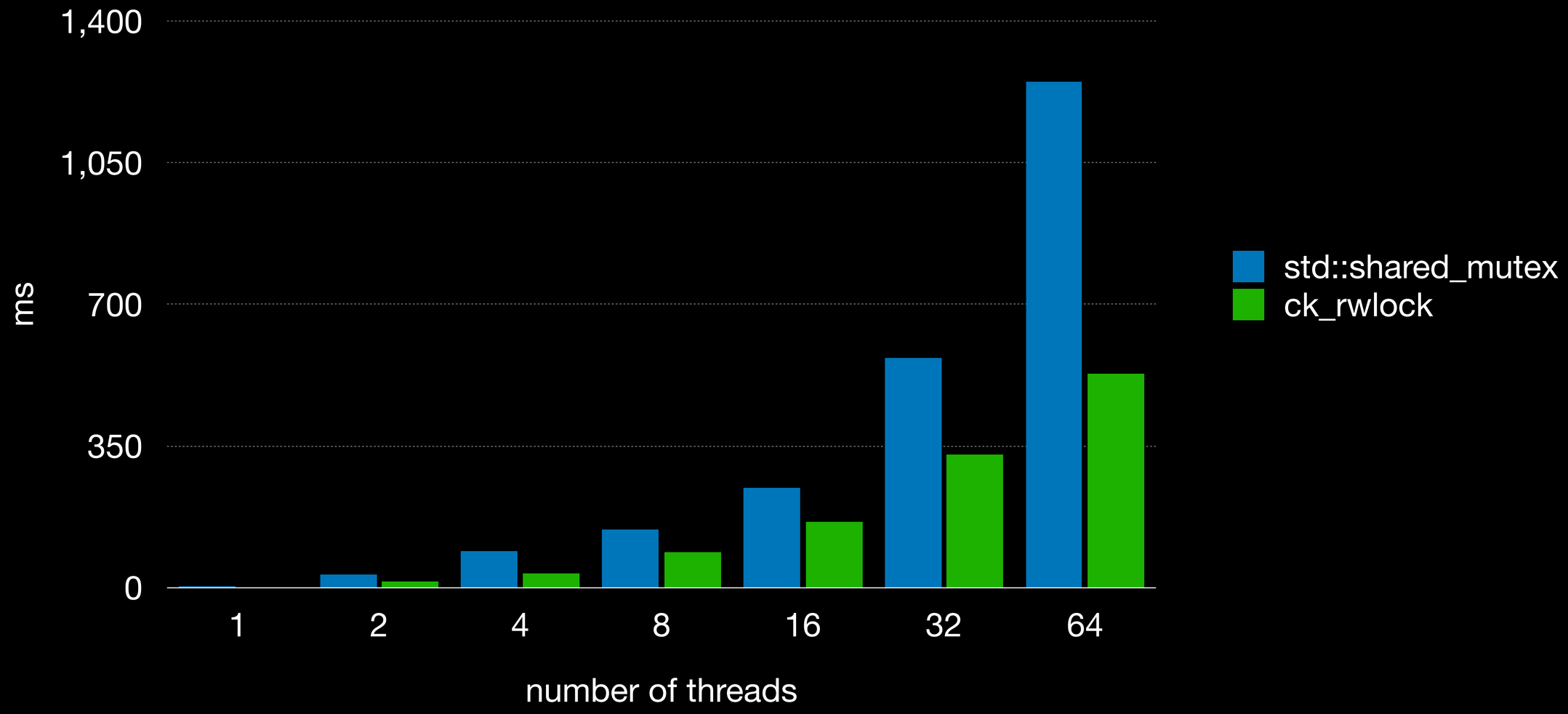
(completion time of 10,000 read op on each thread, lower is better)



Micro-Benchmark of ck_rwlock

read-read performance

(completion time of 10,000 read op on each thread, lower is better)

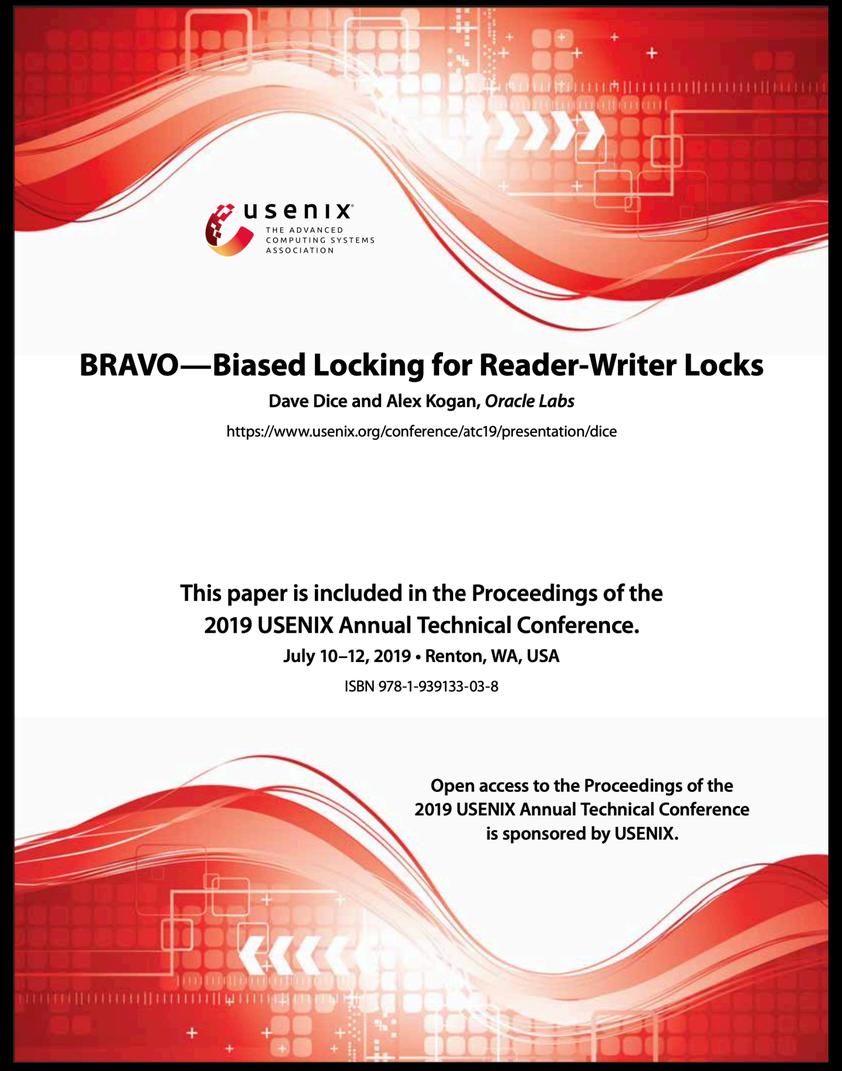


BRAVO - Biased Locking for Reader-Writer Locks

Dave Dice and Alen Kogan, Oracle Labs - USENIX'19

> BRAVO acts as an accelerator layer, as readers can always
> fall back to the traditional underlying lock to gain read access.
> ...
> Write performance and the scalability of read-vs-write and
> write-vs-write behavior depends solely on the underlying lock.

- Section 3.



BRAVO implementation in C++ (#9394)

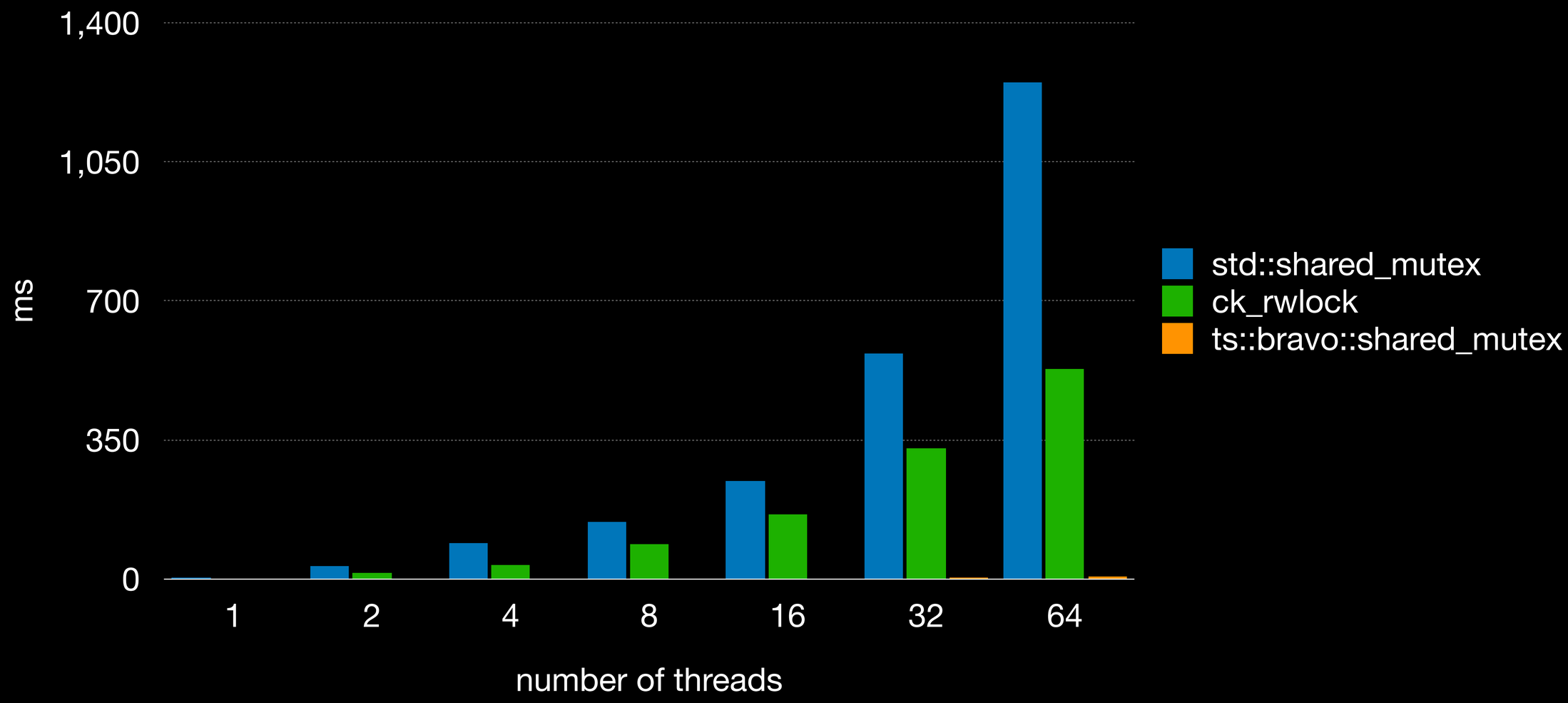
```
template <typename T = std::shared_mutex, size_t SLOT_SIZE = 25
class shared_mutex_impl
{
  struct alignas(hardware_constructive_interference_size) Slot
    std::atomic<bool> mu = false;
  };

  struct Mutex {
    std::atomic<bool> read_bias          = false;
    std::array<Slot, SLOT_SIZE> readers = {};
    time_point inhibit_until{};
    T underlying;
  };
```

```
void
lock_shared(Token &token)
{
  // Fast path
  if (_mutex.read_bias.load(std::memory_order_acquire)) {
    size_t index = DenseThreadId::self() % SLOT_SIZE;
    Slot &slot   = _mutex.readers[index];
    bool expect  = false;
    if (slot.mu.compare_exchange_strong(expect, true, std::memo
      // recheck
      if (_mutex.read_bias.load(std::memory_order_acquire)) {
        token = index + 1;
        return;
      } else {
        slot.mu.store(false, std::memory_order_relaxed);
      }
    }
  }

  // Slow path
  _mutex.underlying.lock_shared();
  if (_mutex.read_bias.load(std::memory_order_acquire) == false
    _mutex.read_bias.store(true, std::memory_order_release);
  }
}
```

Micro-Benchmark of BRAVO Implementation

read-read performance

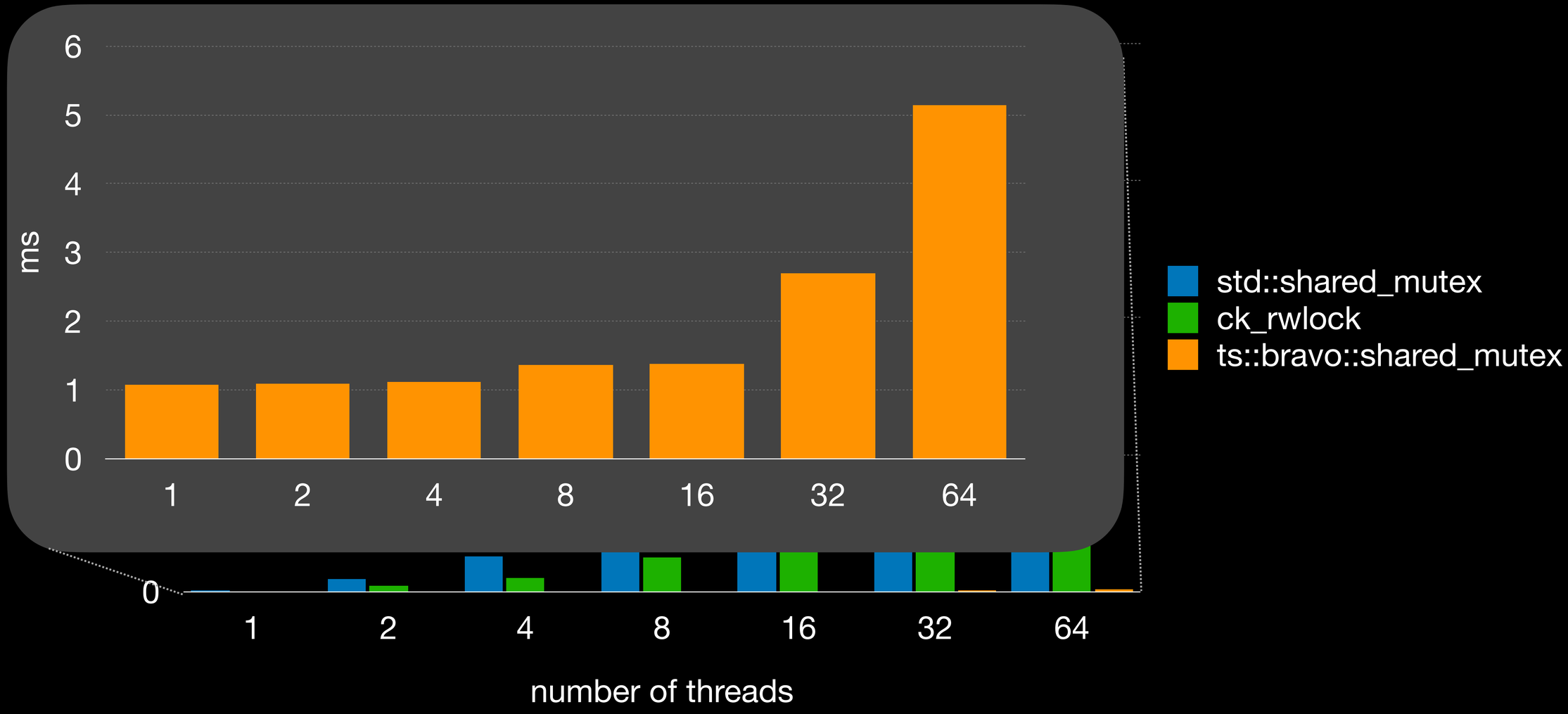
(completion time of 10,000 read op on each thread, lower is better)



Micro-Benchmark of BRAVO Implementation

read-read performance

(completion time of 10,000 read op on each thread, lower is better)



BRAVO Reader-Writer Lock Use Case

and candidates

- librecords (ink_rwlock)
  - Use BRAVO lock for g_records_rwlock #9395
  - 5% improvement (554,388 -> 582,533 rps)

- HostDB (std::shared_mutex)
  - Introduced by Replace exclusive locks with rwlocks in hostdb #9442

- CacheHostTable (std::shared_mutex)
  - Introduced by Fix hosting.config reload #9046
  - std::shared_ptr might be better approach

TrafficServer (9.2.0)
+ Replace Vol Mutex with std::shared_mutex
+ BRAVO reader-writer lock (#9394)

Rule #1. No write op under reader lock

RAM Cache

Under holding reader lock

P_CacheVol.h

```
123 struct Vol : public Continuation {
...
155   RamCache *ram_cache = nullptr;
```

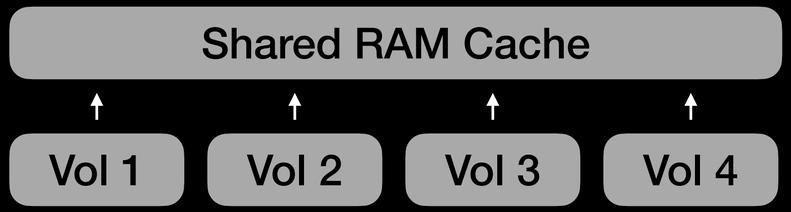
RamCacheLRU.cc

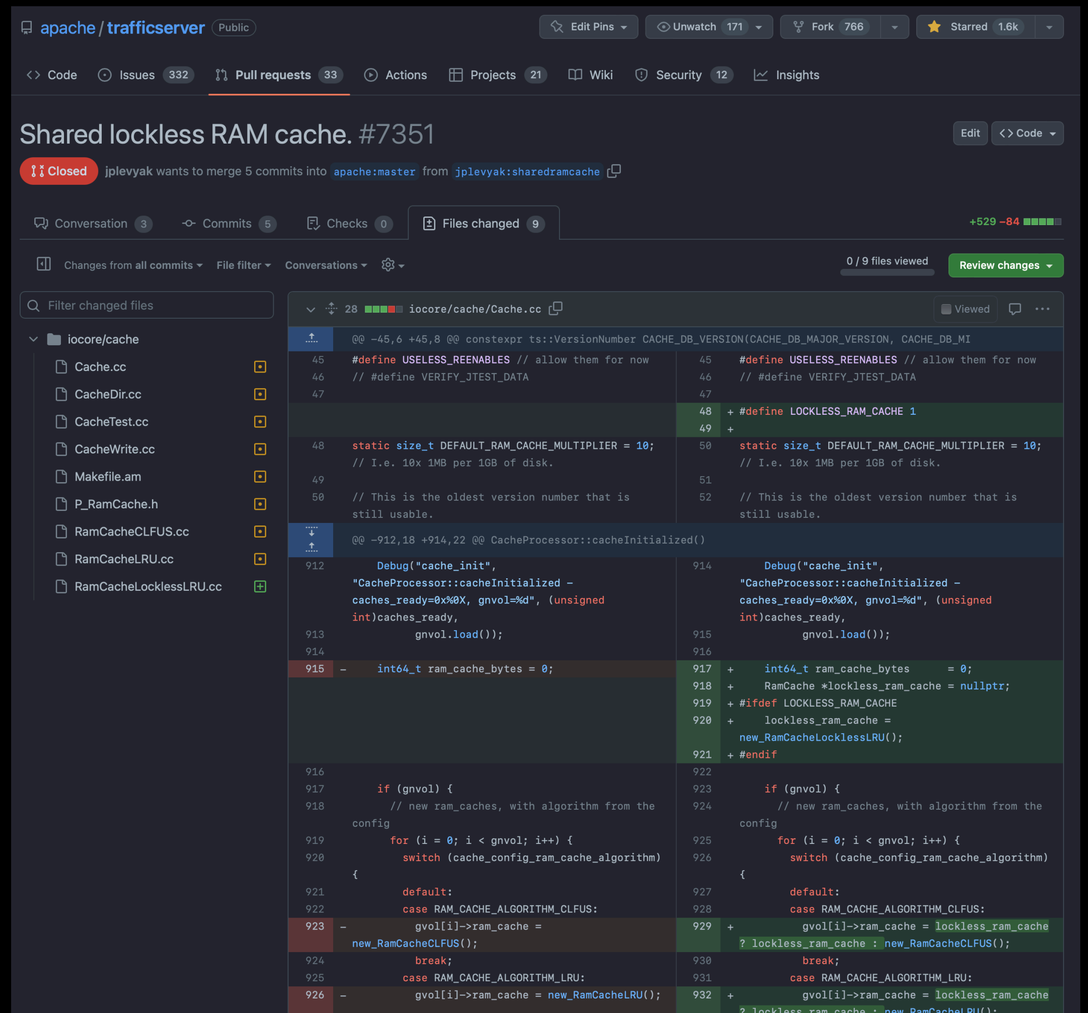
```
119 int
120 RamCacheLRU::get(CryptoHash *key, Ptr<IOBufferData> *ret_data, uint64_t auxkey)
121 {
...
125   uint32_t i           = key->slice32(3) % nbuckets;
126   RamCacheLRUEntry *e = bucket[i].head;
127   while (e) {
128     if (e->key == *key && e->auxkey == auxkey) {
129       lru.remove(e);
130       lru.enqueue(e);
131       (*ret_data) = e->data;
132       DDebug("ram_cache", "get %X %" PRIu64 " HIT", key->slice32(3), auxkey);
133       CACHE_SUM_DYN_STAT_THREAD(cache_ram_cache_hits_stat, 1);
134       return 1;
135     }
136     e = e->hash_link.next;
137   }
```

Shared Lockless RAM Cache (#7351)

by John Plevyak <jplevyak@apache.org>

- Shared RAM Cache
- LRU with atomic operations

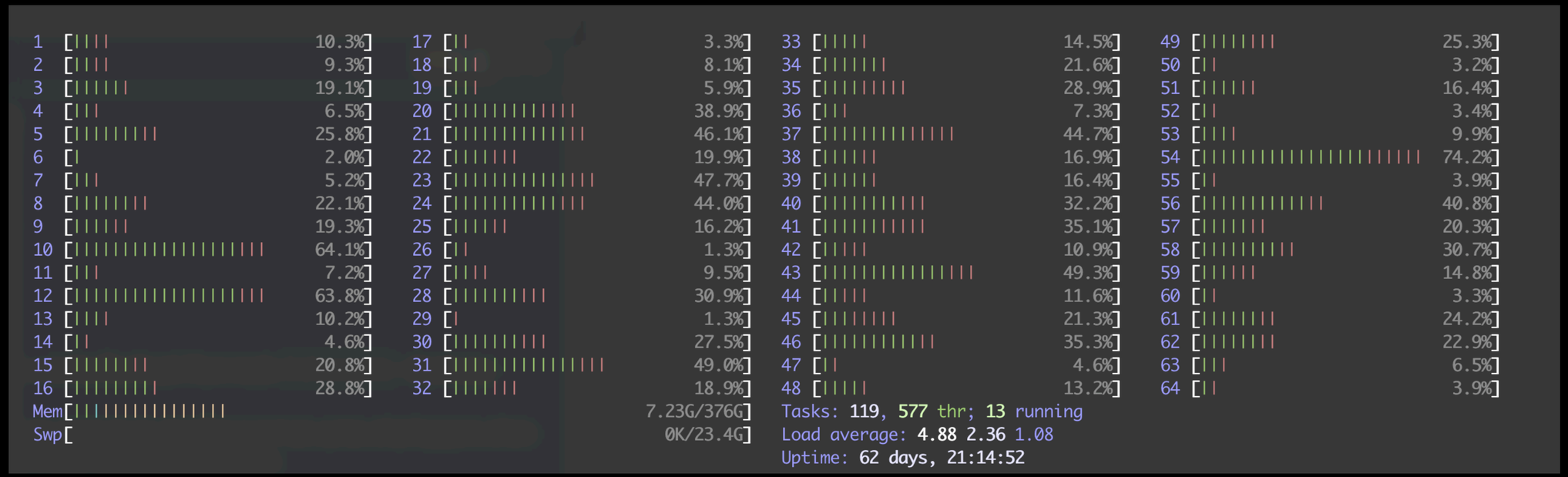




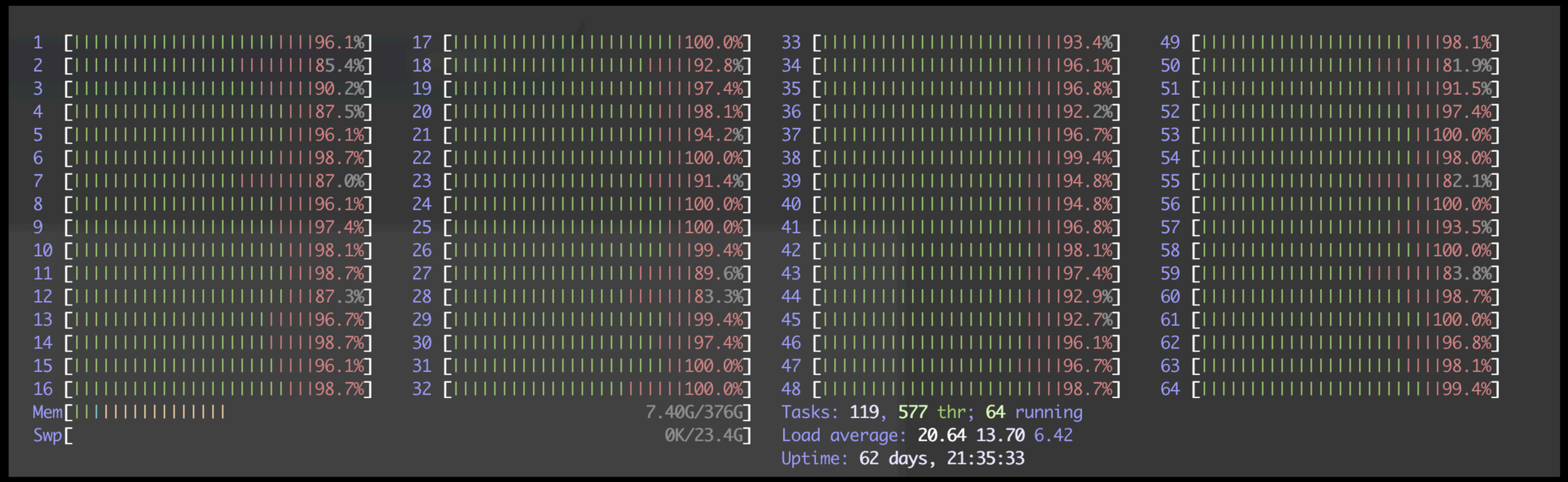
TrafficServer (9.2.0)
+ Replace Vol Mutex with std::shared_mutex
+ BRAVO reader-writer lock (#9394)
+ Shared Lockless RAM Cache (#7351)

64 threads

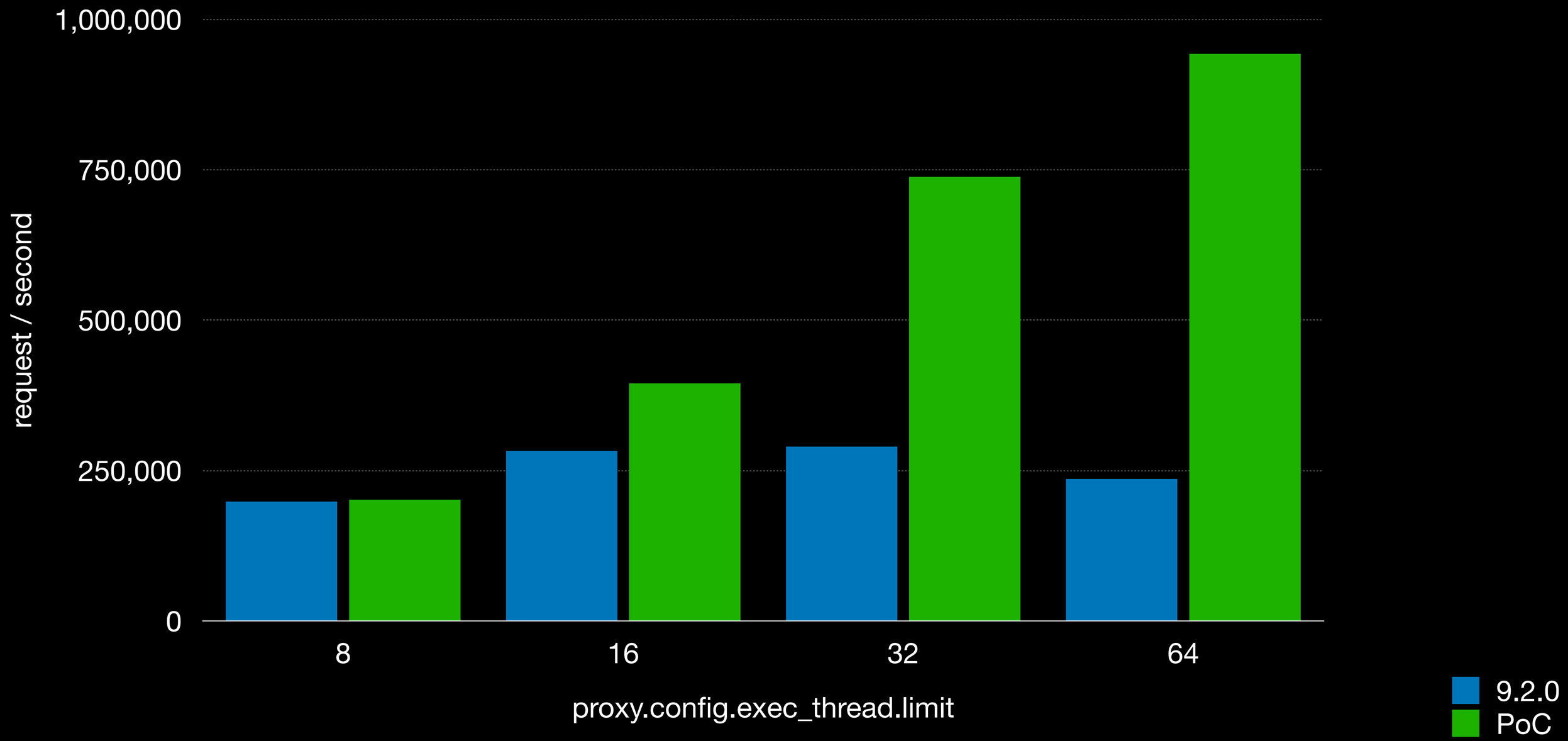
9.2.0



PoC

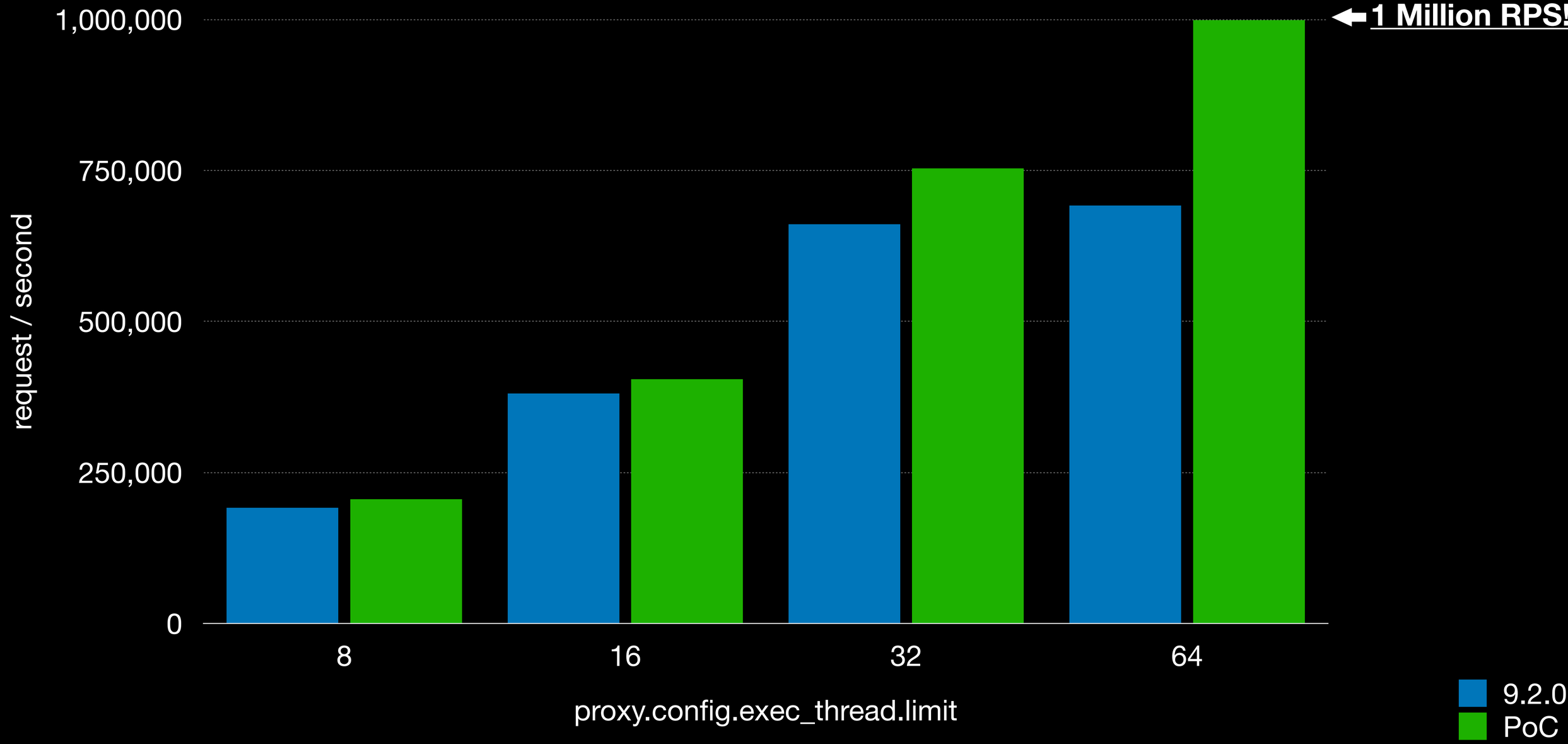


URL=1, Content Size=0B



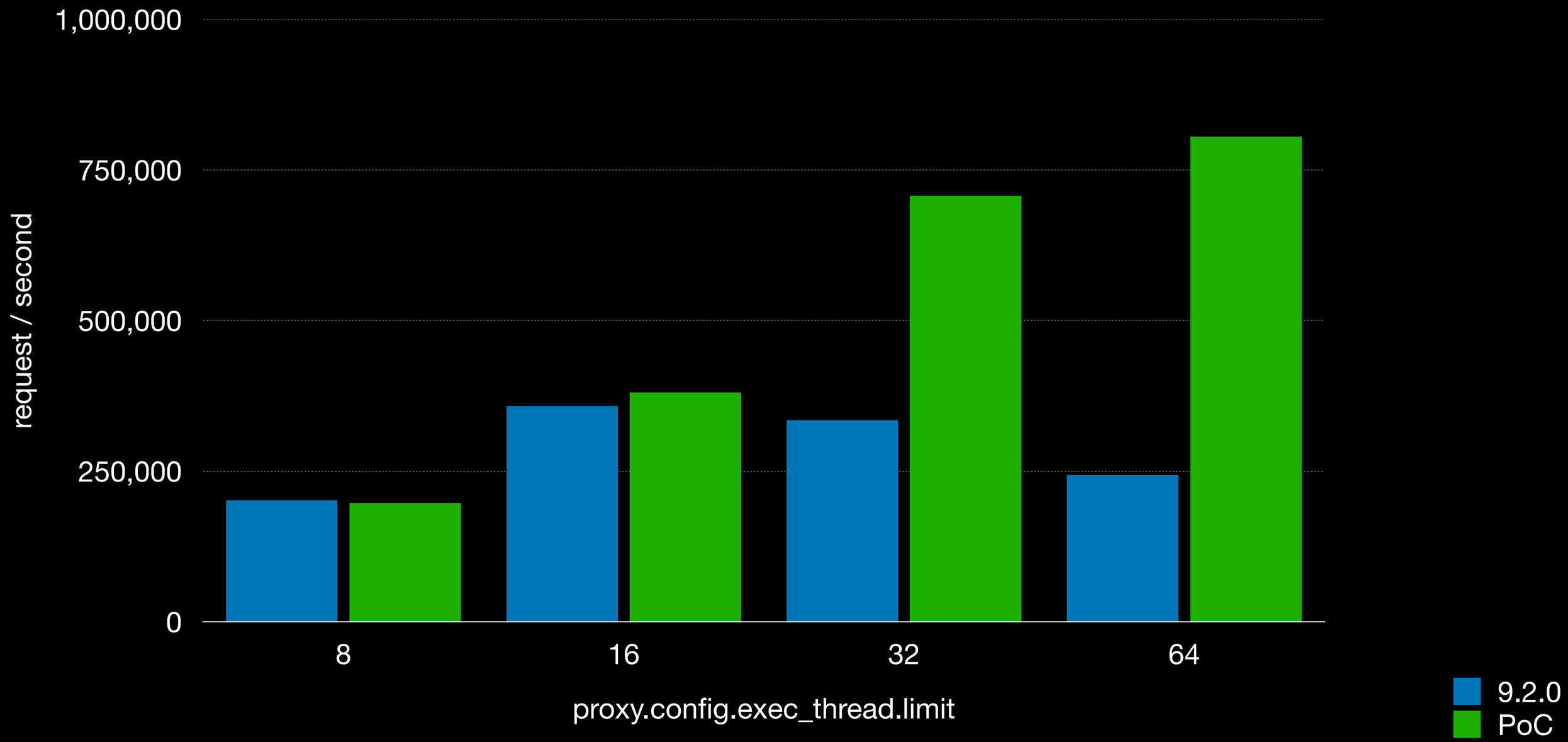
Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz 64 Core

URL=64, Content Size=0B, volume=10



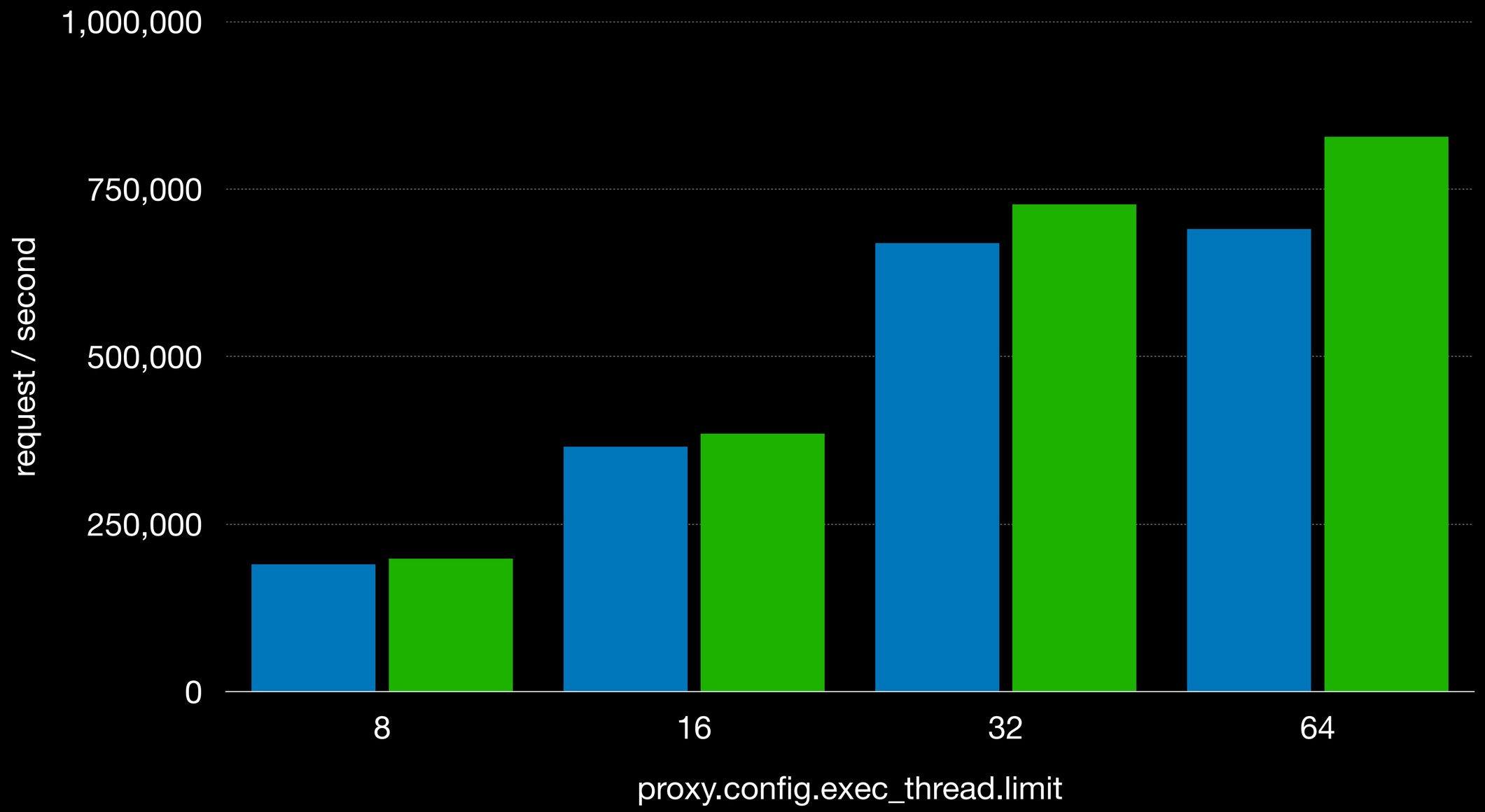
Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz 64 Core

URL=1, Content Size=1KB



Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz 64 Core

URL=64, Content Size=1KB, volume=10



Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz 64 Core

Can you deploy on production!?

Challenges

Adjust current code with reader lock

CacheRead.cc

```
524 int
525 CacheVC::openReadClose(int event, Event * /* e ATS_UNUSED */)
525 {
...
534   CACHE_TRY_LOCK(lock, vol->mutex, mutex->thread_holding);
535   if (!lock.is_locked()) {
536     VC_SCHED_LOCK_RETRY();
537   }
538   if (f.hit_evacuate && dir_valid(vol, &first_dir) && closed > 0) {
539     if (f.single_fragment) {
540       vol->force_evacuate_head(&first_dir, dir_pinned(&first_dir));
541     } else if (dir_valid(vol, &earliest_dir)) {
542       vol->force_evacuate_head(&first_dir, dir_pinned(&first_dir));
543       vol->force_evacuate_head(&earliest_dir, dir_pinned(&earliest_dir));
544     }
545   }
```

Challenges

Adjust current code with reader lock

CacheDir.cc

```
535 int
536 dir_probe(const CacheKey *key, Vol *vol, Dir *result, Dir **last_collision)
537 {
...
549   e = dir_bucket(b, seg);
...
570         if (dir_valid(vol, e)) {
...
577         } else { // delete the invalid entry
578           CACHE_DEC_DIR_USED(vol->mutex);
579           e = dir_delete_entry(e, p, s, vol);
580           continue;
581         }
```

Summary

- PoC: TrafficServer (9.2.0)
  + Replace Vol Mutex with std::shared_mutex
  + BRAVO reader-writer lock (#9394)
  + Shared Lockless RAM Cache (#7351)

- More works to adjust code with reader lock is required

- BRAVO reader-writer lock can apply read heavy cases

Apache Traffic Server

Related Issue

cache_dir_sync triggers an increase in cache read time on ATS 9.1.3 #9124

- Root cause is Try Lock Contention of Vol Mutex
- Increasing number of volumes is mitigation (1 -> 5)

CacheDir.cc

```
1072 int
1073 CacheSync::mainEvent(int event, Event *e)
1074 {
 ...
1117     CACHE_TRY_LOCK(lock, gvol[vol_idx]->mutex, mutex->thread_holding);
1118     if (!lock.is_locked()) {
1119       trigger = eventProcessor.schedule_in(this, HRTIME_MSECONDS(cache_config_mutex_retry_
1120       return EVENT_CONT;
1121     }
```

Benchmark Conditions

Client

- Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz 64 Core
- wrk2, 64 threads

```
» taskset -c 0-63 wrk2 -t 64 -c 640 -d 10 -R 2000000 http://targetbox/static/0B

» taskset -c 0-63 wrk2 -t 64 -c 640 -d 10 -R 2000000 http://targetbox/static/1KB
```

Benchmark Conditions

ATS 9.2.0

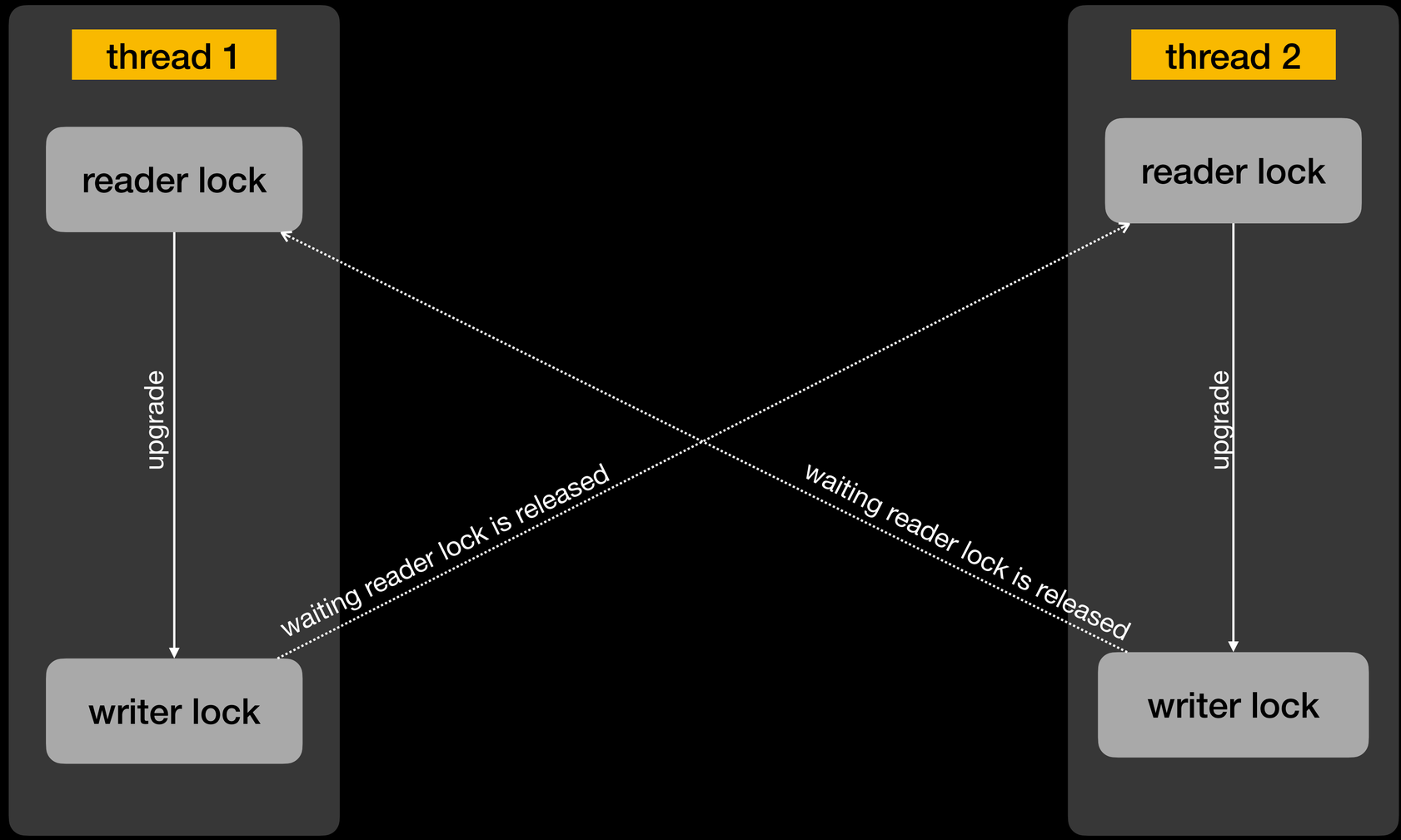
- Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz 64 Core
- jemalloc 5.3.0

```
» traffic_server -fF
```

- records.config

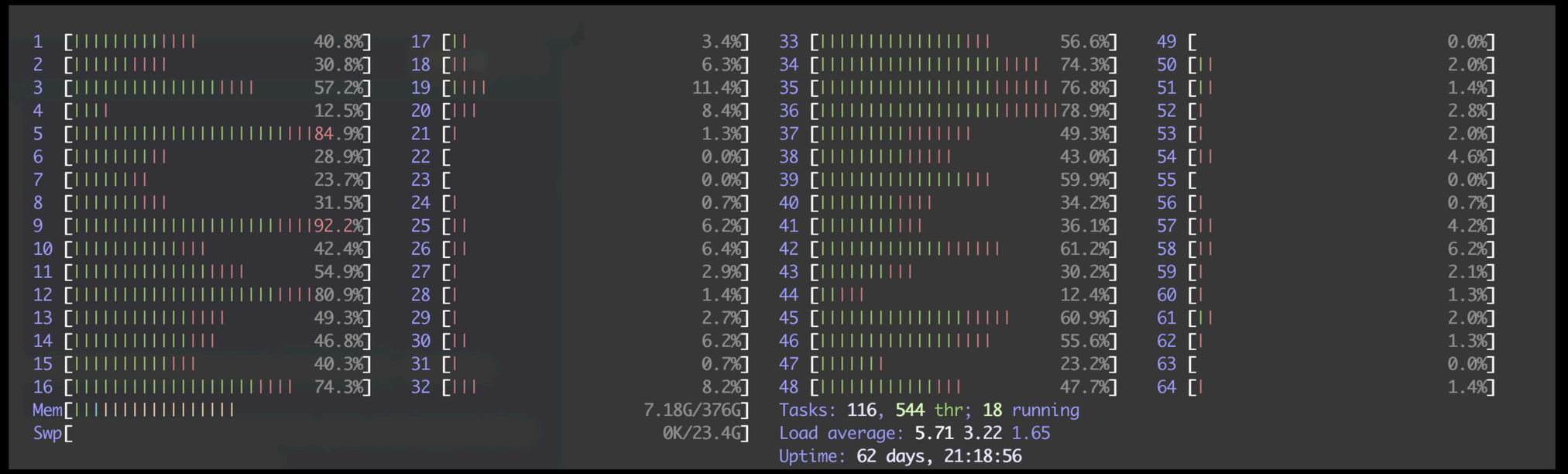
```
CONFIG proxy.config.accept_threads INT 0
CONFIG proxy.config.exec_thread.listen INT 1
CONFIG proxy.config.exec_thread.autoconfig INT 0
CONFIG proxy.config.exec_thread.limit INT 64            # 0,1,2,4,8,16,32,64
CONFIG proxy.config.exec_thread.affinity INT 4

CONFIG proxy.config.cache.ram_cache.size INT 104857600  # 100MB

CONFIG proxy.config.http.insert_response_via_str INT 2
```

Dead-lock of upgrading



32 threads

9.2.0



PoC