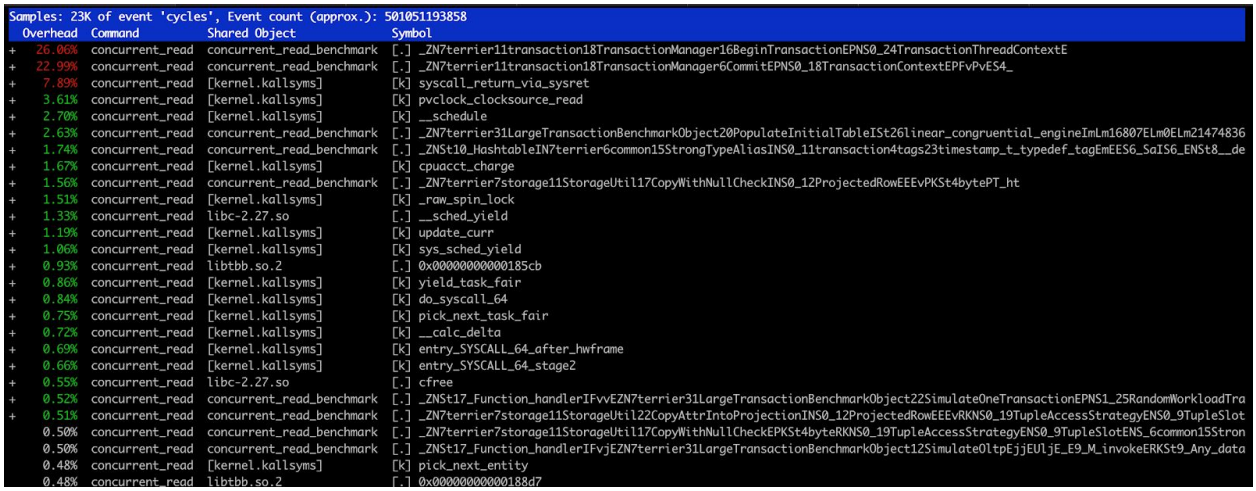
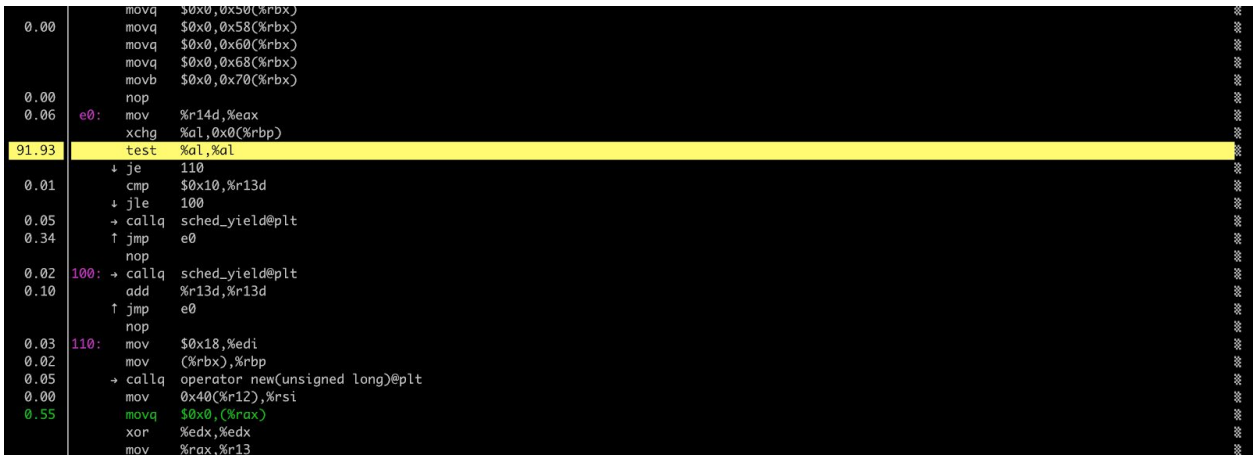
Original Hotspots detected



```
Samples: 23K of event 'cycles', Event count (approx.): 501051193858
  Overhead  Command          Shared Object                Symbol
+   26.06%  concurrent_read  concurrent_read_benchmark    [.] _ZN7terrier11transaction18TransactionManager16BeginTransactionEPNS0_24TransactionThreadContextE
+   22.99%  concurrent_read  concurrent_read_benchmark    [.] _ZN7terrier11transaction18TransactionManager6CommitEPNS0_18TransactionContextEPFvPvES4_
+    7.89%  concurrent_read  [kernel.kallsyms]            [k] syscall_return_via_sysret
+    3.61%  concurrent_read  [kernel.kallsyms]            [k] pvclock_clocksource_read
+    2.70%  concurrent_read  [kernel.kallsyms]            [k] __schedule
+    2.63%  concurrent_read  concurrent_read_benchmark    [.] _ZN7terrier31LargeTransactionBenchmarkObject20PopulateInitialTableISt26linear_congruential_engineImLm16807ELm0ELm21474836
+    1.74%  concurrent_read  concurrent_read_benchmark    [.] _ZNSt10_HashtableIN7terrier6common15StrongTypeAliasINS0_11transaction4tags23timestamp_t_typedef_tagEmEES6_SaIS6_ENSt8__de
+    1.67%  concurrent_read  [kernel.kallsyms]            [k] cpuacct_charge
+    1.56%  concurrent_read  concurrent_read_benchmark    [.] _ZN7terrier7storage11StorageUtil17CopyWithNullCheckINS0_12ProjectedRowEEEvPKSt4bytePT_ht
+    1.51%  concurrent_read  [kernel.kallsyms]            [k] _raw_spin_lock
+    1.33%  concurrent_read  libc-2.27.so                 [.] __sched_yield
+    1.19%  concurrent_read  [kernel.kallsyms]            [k] update_curr
+    1.06%  concurrent_read  [kernel.kallsyms]            [k] sys_sched_yield
+    0.93%  concurrent_read  libtbb.so.2                  [.] 0x00000000000185cb
+    0.86%  concurrent_read  [kernel.kallsyms]            [k] yield_task_fair
+    0.84%  concurrent_read  [kernel.kallsyms]            [k] do_syscall_64
+    0.75%  concurrent_read  [kernel.kallsyms]            [k] pick_next_task_fair
+    0.72%  concurrent_read  [kernel.kallsyms]            [k] __calc_delta
+    0.69%  concurrent_read  [kernel.kallsyms]            [k] entry_SYSCALL_64_after_hwframe
+    0.66%  concurrent_read  [kernel.kallsyms]            [k] entry_SYSCALL_64_stage2
+    0.55%  concurrent_read  libc-2.27.so                 [.] cfree
+    0.52%  concurrent_read  concurrent_read_benchmark    [.] _ZNSt17_Function_handlerIFvvEZN7terrier31LargeTransactionBenchmarkObject22SimulateOneTransactionEPNS1_25RandomWorkloadTra
+    0.51%  concurrent_read  concurrent_read_benchmark    [.] _ZN7terrier7storage11StorageUtil22CopyAttrIntoProjectionINS0_12ProjectedRowEEEvRKNS0_19TupleAccessStrategyENS0_9TupleSlot
     0.50%  concurrent_read  concurrent_read_benchmark    [.] _ZN7terrier7storage11StorageUtil17CopyWithNullCheckEPKSt4byteRKNS0_19TupleAccessStrategyENS0_9TupleSlotENS_6common15Stron
     0.50%  concurrent_read  concurrent_read_benchmark    [.] _ZNSt17_Function_handlerIFvjEZN7terrier31LargeTransactionBenchmarkObject12SimulateOltpEjjEUljE_E9_M_invokeERKSt9_Any_data
     0.48%  concurrent_read  [kernel.kallsyms]            [k] pick_next_entity
     0.48%  concurrent_read  libtbb.so.2                  [.] 0x00000000000188d7
```

The above image shows that BeginTransaction and Commit are the major contention hotspots in the TransactionManager.

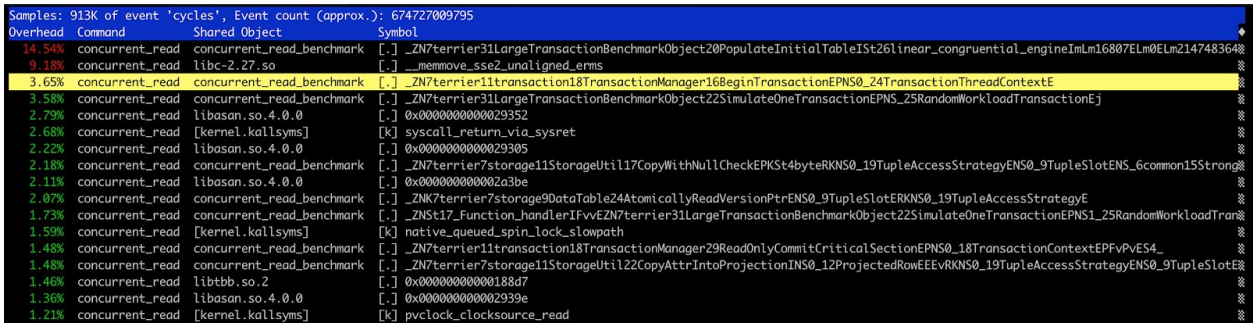
Annotated BeginTransaction



```
                movq   $0x0,0x50(%rbx)
  0.00          movq   $0x0,0x58(%rbx)
                movq   $0x0,0x60(%rbx)
                movq   $0x0,0x68(%rbx)
                movb   $0x0,0x70(%rbx)
  0.00          nop
  0.06   e0:    mov    %r14d,%eax
                xchg   %al,0x0(%rbp)
 91.93          test   %al,%al
              ↓ je     110
  0.01          cmp    $0x10,%r13d
              ↓ jle    100
  0.05        → callq  sched_yield@plt
  0.34        ↑ jmp    e0
                nop
  0.02  100:  → callq  sched_yield@plt
  0.10          add    %r13d,%r13d
              ↑ jmp    e0
                nop
  0.03  110:    mov    $0x18,%edi
  0.02          mov    (%rbx),%rbp
  0.05        → callq  operator new(unsigned long)@plt
  0.00          mov    0x40(%r12),%rsi
  0.55          movq   $0x0,(%rax)
                xor    %edx,%edx
                mov    %rax,%r13
```

This image shows the annotated instructions hotspot in the BeginTransaction function. We can see that the ‘test %al %al’ instruction takes up almost 92% of the execution time. If it doesn’t succeed, a call is being made to ‘sched_yield’ which makes the thread to relinquish CPU. From the source code, I’ve seen that a single global latch ‘curr_running_txns_latch_’ is being used in both BeginTransaction and Commit, which can be replaced with per-thread data-structures and latches.

Perf report after the fix



```
Samples: 913K of event 'cycles', Event count (approx.): 674727009795
Overhead  Command          Shared Object              Symbol
  14.54%  concurrent_read  concurrent_read_benchmark  [.] _ZN7terrier31LargeTransactionBenchmarkObject20PopulateInitialTableISt26linear_congruential_engineImLm16807ELm0ELm2147483647
   9.18%  concurrent_read  libc-2.27.so              [.] __memmove_sse2_unaligned_erms
   3.65%  concurrent_read  concurrent_read_benchmark  [.] _ZN7terrier11transaction18TransactionManager16BeginTransactionEPNS0_24TransactionThreadContextE
   3.58%  concurrent_read  concurrent_read_benchmark  [.] _ZN7terrier31LargeTransactionBenchmarkObject22SimulateOneTransactionEPNS_25RandomWorkloadTransactionEj
   2.79%  concurrent_read  libasan.so.4.0.0           [.] 0x0000000000029352
   2.68%  concurrent_read  [kernel.kallsyms]          [k] syscall_return_via_sysret
   2.22%  concurrent_read  libasan.so.4.0.0           [.] 0x0000000000029305
   2.18%  concurrent_read  concurrent_read_benchmark  [.] _ZN7terrier7storage11StorageUtil17CopyWithNullCheckEPKSt4byteRKNS0_19TupleAccessStrategyENS0_9TupleSlotENS_6common15Strong
   2.11%  concurrent_read  libasan.so.4.0.0           [.] 0x000000000002a3be
   2.07%  concurrent_read  concurrent_read_benchmark  [.] _ZNK7terrier7storage9DataTable24AtomicallyReadVersionPtrENS0_9TupleSlotERKNS0_19TupleAccessStrategyE
   1.73%  concurrent_read  concurrent_read_benchmark  [.] _ZNSt17_Function_handlerIFvvEZN7terrier31LargeTransactionBenchmarkObject22SimulateOneTransactionEPNS1_25RandomWorkloadTran
   1.59%  concurrent_read  [kernel.kallsyms]          [k] native_queued_spin_lock_slowpath
   1.48%  concurrent_read  concurrent_read_benchmark  [.] _ZN7terrier11transaction18TransactionManager29ReadOnlyCommitCriticalSectionEPNS0_18TransactionContextEPFvPvES4_
   1.48%  concurrent_read  concurrent_read_benchmark  [.] _ZN7terrier7storage11StorageUtil22CopyAttrIntoProjectionINS0_12ProjectedRowEEEvRKNS0_19TupleAccessStrategyENS0_9TupleSlotE
   1.46%  concurrent_read  libtbb.so.2                [.] 0x00000000000188d7
   1.36%  concurrent_read  libasan.so.4.0.0           [.] 0x000000000002939e
   1.21%  concurrent_read  [kernel.kallsyms]          [k] pvclock_clocksource_read
```

The BeginTransaction and Commit are no longer the major bottlenecks in the system.

The new bottleneck in BeginTransaction

```
         ↓ jne    e2e
           mov    %rdi,-0x380(%rbp)
  0.11   → callq  tbb::interface5::reader_writer_lock::lock_read()@plt
  0.13     lea    0x8(%r13),%rdi
  0.00     mov    %rdi,%rax
  0.00     shr    $0x3,%rax
  0.00     cmpb   $0x0,0x7fff8000(%rax)
  0.12   ↓ jne    f56
  0.06     mov    $0x1,%eax
           lock   xadd   %rax,(%rdi)
 38.23     mov    (%rsp),%rsi
  0.13     lea    0x320(%rsi),%rdi
           mov    %rdi,%rdx
           shr    $0x3,%rdx
  0.03     cmpb   $0x0,0x7fff8000(%rdx)
  0.06   ↓ jne    f51
  0.00     lea    0x1a0(%rsi),%rdi
           mov    %rax,-0x1c0(%rbp)
  0.02     movb   $0xf8,0x7fff8000(%rdx)
  0.07     mov    %rdi,%rsi
           shr    $0x3,%rsi
           cmpb   $0x0,0x7fff8000(%rsi)
  0.02   ↓ jne    f4c
  0.05     mov    (%rsp),%rdx
```

Implementation analysis

The existing code was using a global data-structure to maintain the list of all running and completed transactions across all workers and hence was using a global lock to secure access to these lists.

Instead of this, we can maintain this list of completed and running transactions inside the TransactionThreadContext and have per-thread locks. This will greatly reduce the bottleneck of locking in BeginTransaction, Commit and Abort. Although, CompletedTransactionsForGC and OldestTransactionTimestamp functions still need access to all the transactions.

CompletedTransactionsForGC has been changed to accumulate the completed transactions from one thread at a time into the global completed transactions list. This prevents the need to lock all the threads from beginning or committing transactions.

Similarly, OldestTransactionTimestamp can also be calculated with 1 thread at a time and then choosing the minimum timestamp value across all the threads. If a new transaction gets added to the thread which has been processed in this loop, it’s guaranteed to have greater than the minimum timestamp of the overall transactions since timestamp only increases.