Fourth Annual MySQL Users Conference 2006

Optimizing MySQL on source code level

Vadim Tkachenko
Peter Zaitsev
MySQL AB

Introduction

Vadim Tkachenko

Performance Engineer, MySQL AB

Peter Zaitsev

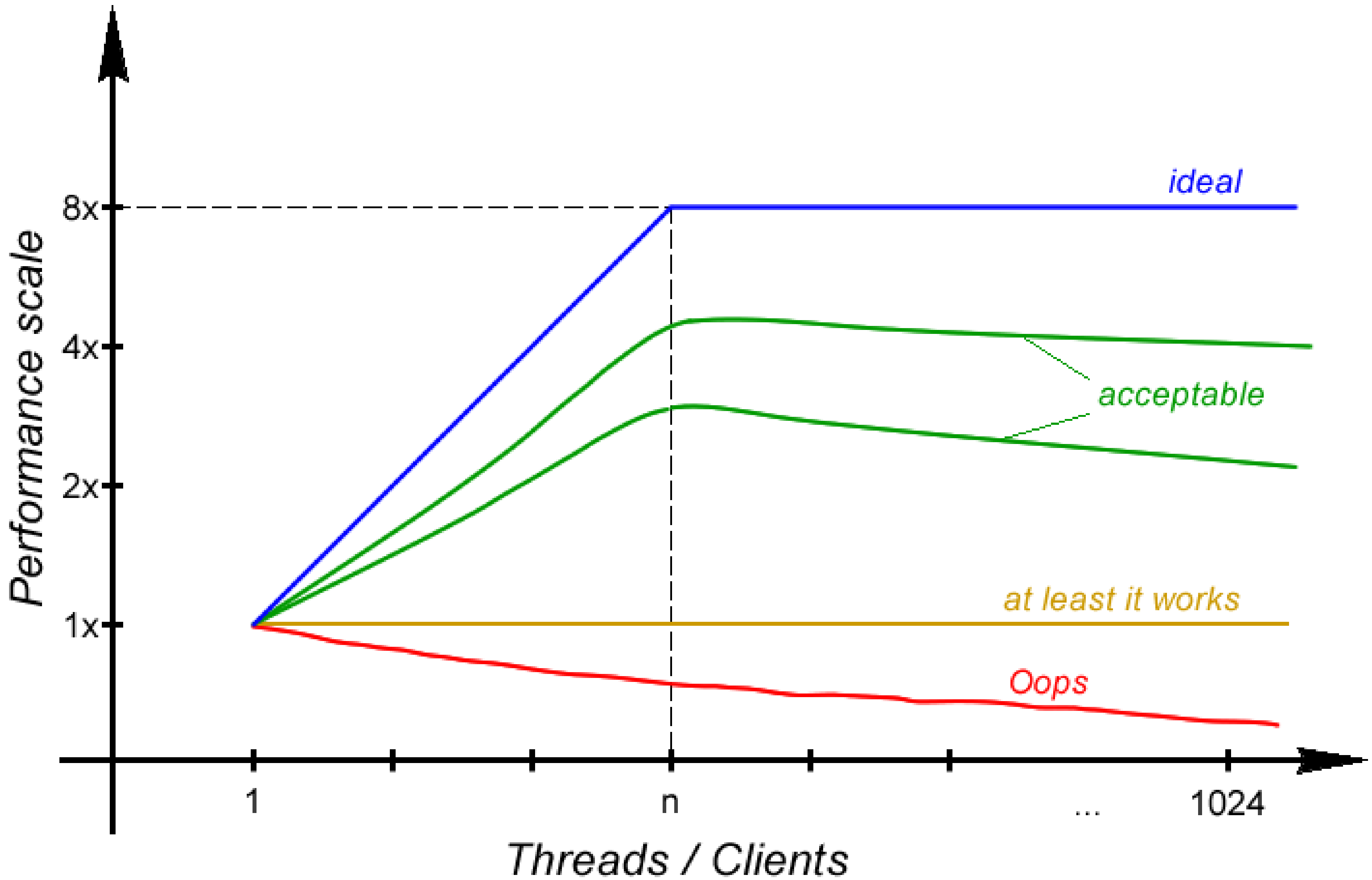
Senior Performance Engineer, MySQL AB

Table of Content

Scalability

- MultiCPU boxes are coming
  - Opteron Dual Core x 2 ways / 4 ways are popular servers
  - Sun T2000 with 32 cores is available
- We are expecting, software on N cpu box
  - At least will work as with 1 cpu
  - In ideal the result will be scaled by N times

Scalability, 8CPU



MyISAM

Wide range-index queries

```
CREATE TABLE `sbtest` (
  `id` int(11) NOT NULL,
  `k` int(10) unsigned NOT NULL default '0',
  `c` char(120) NOT NULL default '',
  `pad` char(60) NOT NULL default '',
  PRIMARY KEY  (`id`),
  KEY `k` (`k`)
) ENGINE=MyISAM;
```

SELECT count(id) FROM test WHERE id BETWEEN n AND n+20000

ID – primary key, key_cache is enough

The result with 4 threads is worse then with 1 threads on 8 CPU box

Digression, tested boxes

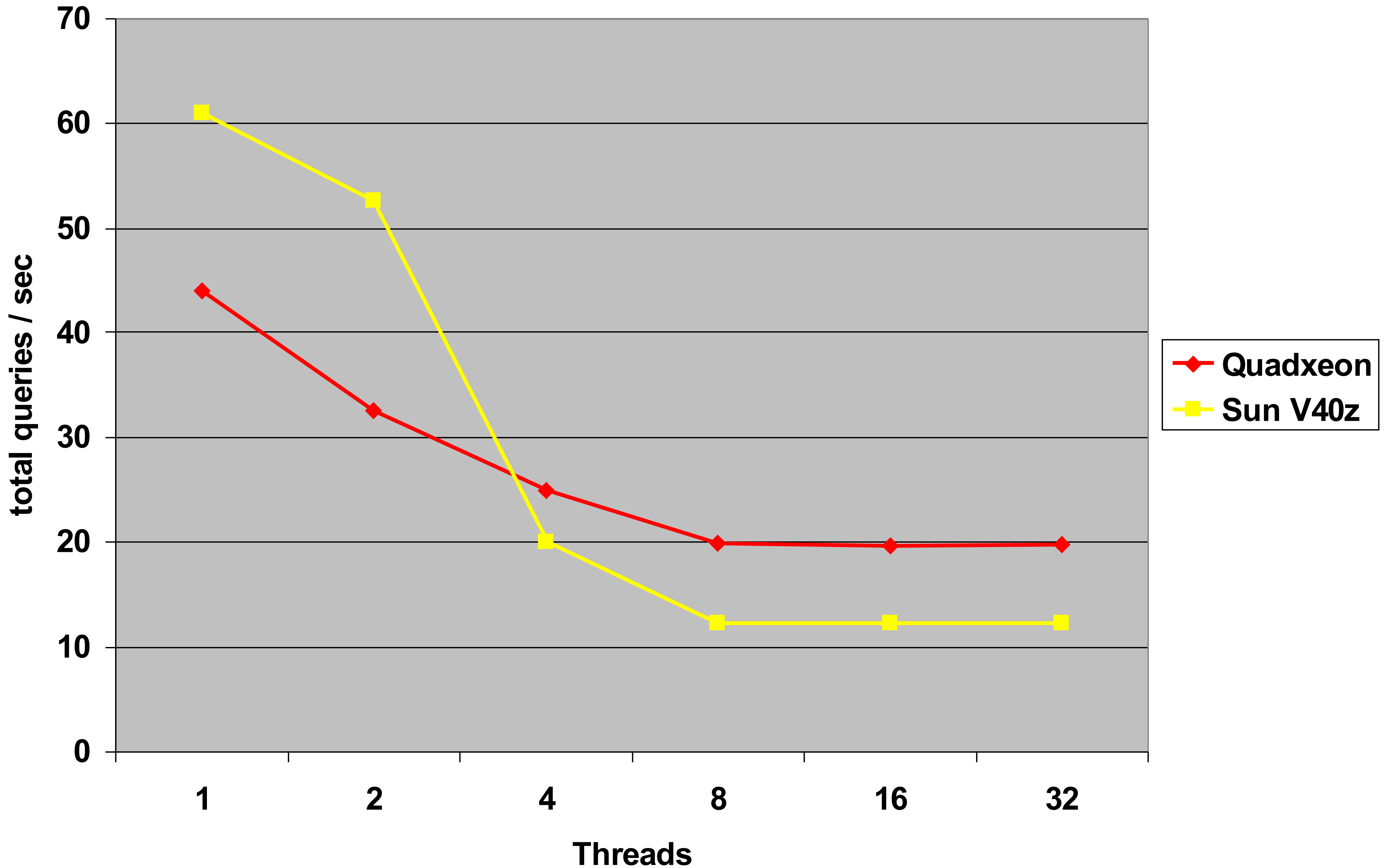
Sun V40z

- Solaris 10
- 4 x Dual Core Opteron @ 2.2GHz (8 logical cpu)
- 16GB of RAM
- StorEdge 3310

Quadxeon

- RedHat AS 3, 2.4.21-15.Elsmp
- 4 x Intel(R) XEON(TM) MP CPU 2.00GHz
- 4GB of RAM
- SATA RAID 10

Initial results



What's wrong

Vmstat / Sun V40z

```
1 thread          |      8 thread
cs us sy id       |   cs    us sy id
475 12  0 87      |   39599 95 4 1
468 12  0 87      |   39854 96 4 1
```

Vmstat / quadxeon

```
  1 thread        |      8 thread
cs us sy id       |    cs us sy id
398 12  0 87      |   4785 13 85  2
378 13  0 87      |   4849 15 85  0
```

High user CPU on Sun V40z and high sys CPU on Quadxeon

Do you have any idea?

Why is user CPU high, but the result does not scale?

Why is there high system CPU on Linux?

Dtrace

dtrace -n 'profile-1000 {@[execname,ustack()] = count()}'

Take probe 1000 times / sec

```
About 90% of probes are in:

libc.so.1`___lwp_mutex_timedlock+0x7
libc.so.1`queue_lock+0x5e
libc.so.1`rwlock_lock+0xc5
libc.so.1`rw_rdlock_impl+0x9f
libc.so.1`pthread_rwlock_rdlock+0x1a
mysqld`mi_rnext+0x28c
mysqld`ha_myisam::index_next()+0x2a
mysqld`handler::read_range_next()+0x3f
mysqld`handler::read_multi_range_next()+0x1d
```

mi_rnext.c

mi_rnext()

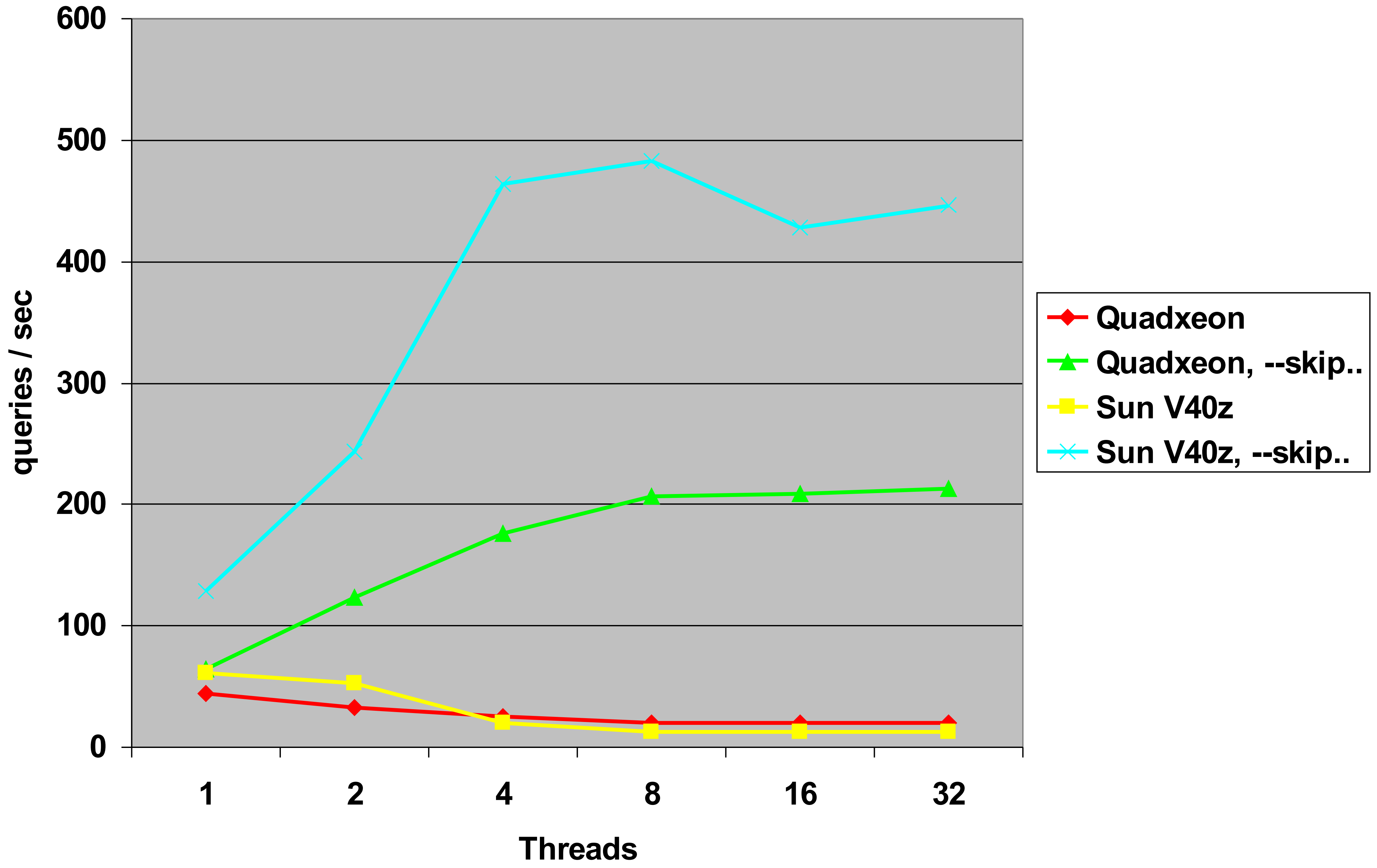
```
…
 if (info->s->concurrent_insert)
    rw_rdlock(&info->s->key_root_lock[inx]);
…
```

Rw_rdlock is called for each row, 20000 times per query

Concurrent insert feature allows to INSERT at the end of file concurrently with SELECT queries

--skip-concurrent-insert disables it

Results, --skip-concurrent…



Why sys time on Linux?

Oprofile

```
  %        kernel function
 28.0434   do_futex
 25.2499   __down_read
 11.9623   unqueue_me
  7.9007   queue_me
  5.8369   hash_futex
```

All functions are rwlock_rdlock related

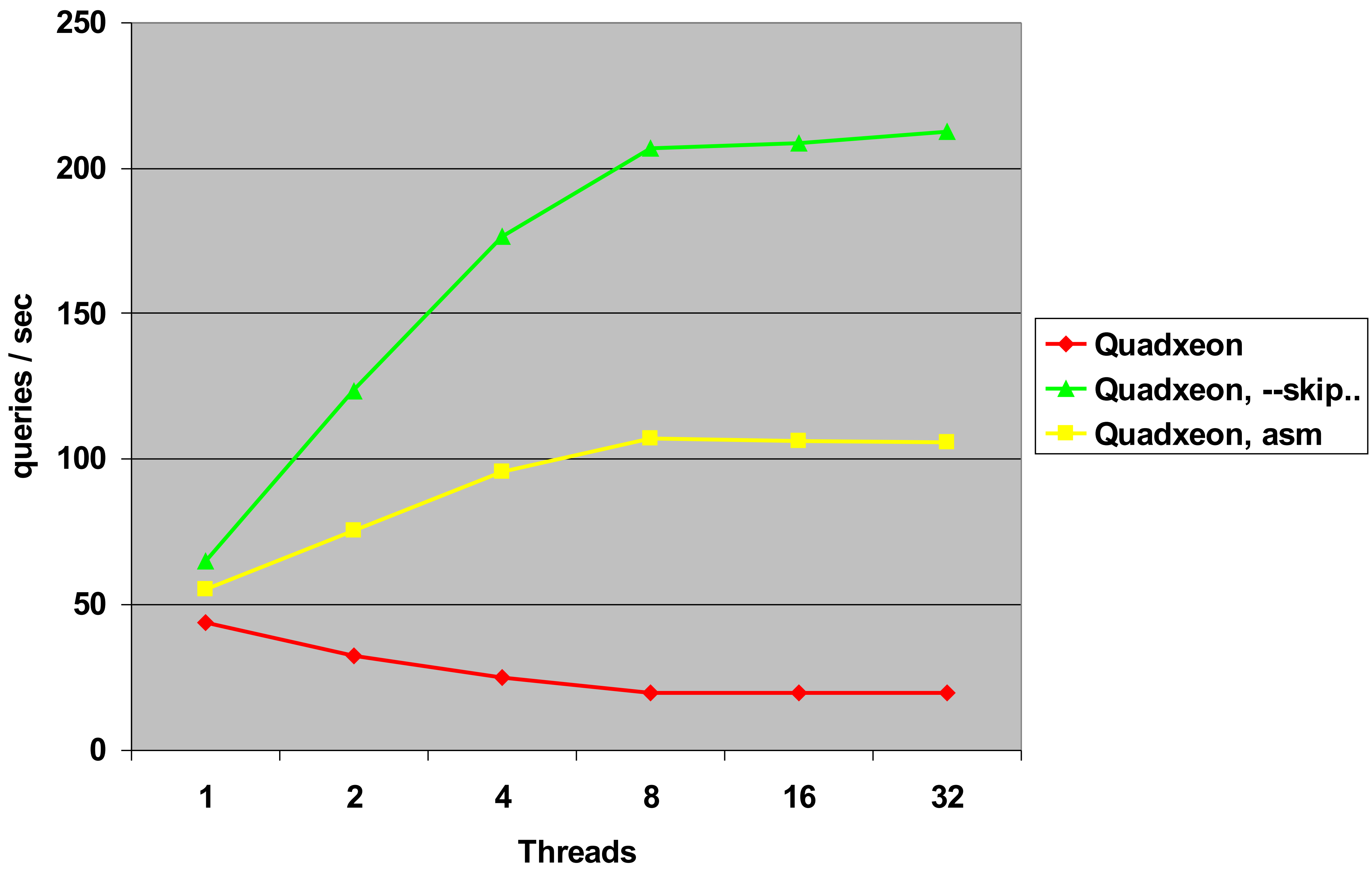
Futex – Fast Userspace Locking

Any futex operation starts in userspace, but it may necessary to communicate with the kernel using the futex(2) system call.

What to do?

- --skip-concurrent-insert, it gives benefit even with 1 thread
- Use pthread_rwlock_rdlock rarely
  - batch read, protect pages of rows, not each row (5.2)
- Use cpu atomic instructions for rw_locks (5.2)
  - Currently only for x86/x86_64 systems
  - The results are not so good as without rw_locks
    - Still contention on memory bus, access to common variable

Results, asm rw_locks



MyISAM, disk read

Wide range-index queries

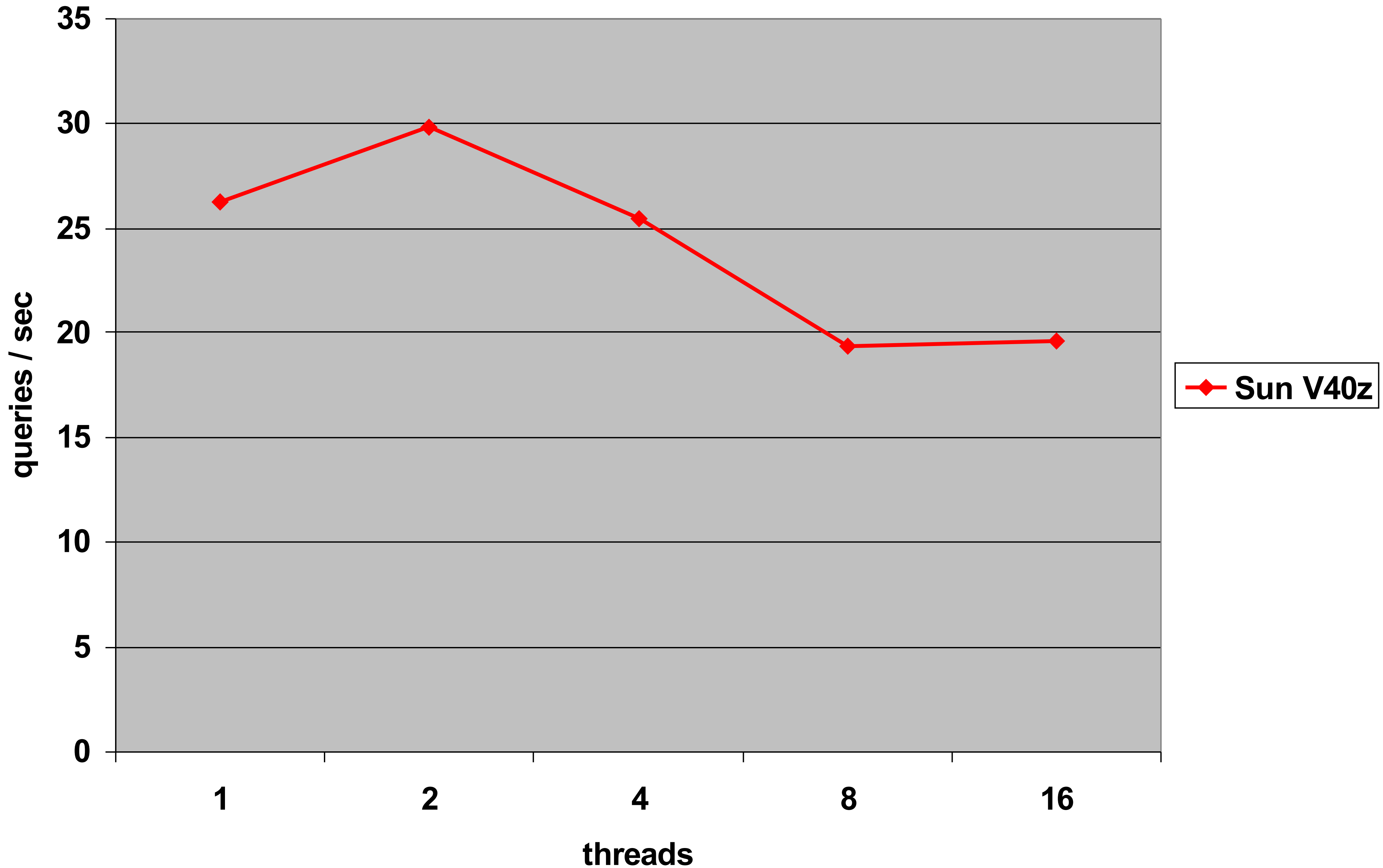
```
CREATE TABLE `sbtest` (
  `id` int(11) NOT NULL,
  `k` int(10) unsigned NOT NULL default '0',
  `c` char(120) NOT NULL default '',
  `pad` char(60) NOT NULL default '',
  PRIMARY KEY  (`id`),
  KEY `k` (`k`)
) ENGINE=MyISAM;
```

SELECT count(c) FROM test WHERE id BETWEEN n AND n+20000

The difference from previous: we read non-indexed column

Let us try it with –skip-concurrent-insert

Initial results



Diagnostic

Vmstat

```
   1 thread                  |          8 threads
 syscall   cs us sy id       | syscall cs us sy id
 499941    218 5  7 87       |   379161 424 4 96  0
 502564    220 5  7 87       |   380220 373 4 96  0
 504734    217 5  7 87       |   380084 383 4 96  0
```

High sys CPU and high numbers of system calls

Do you have any idea why?

Dtrace

```
dtrace -n 'syscall:::entry/pid !=
$pid/{@[execname,probefunc] = count()}'
```

After about 10 sec:

```
Sys call              count
read                    845
lwp_sigmask            1028
gtime                  1297
pread64             3347422
```

Stack:

```
libc.so.1`_pread64+0x7
mysqld`my_pread+0x2c
mysqld`_mi_read_static_record+0x5d
mysqld`mi_rnext+0x25d
mysqld`ha_myisam::index_next(char*)+0x2a
mysqld`handler::read_range_next()+0x3f
mysqld`handler::read_multi_range_next()+0x1d
```

mi_statrec.c

_mi_read_static_record()

```
…
 error=my_pread(info->dfile,(char*) record,info->s->base.reclength,
                  pos,MYF(MY_NABP)) != 0;
…
```

my_pread is macro for pread64

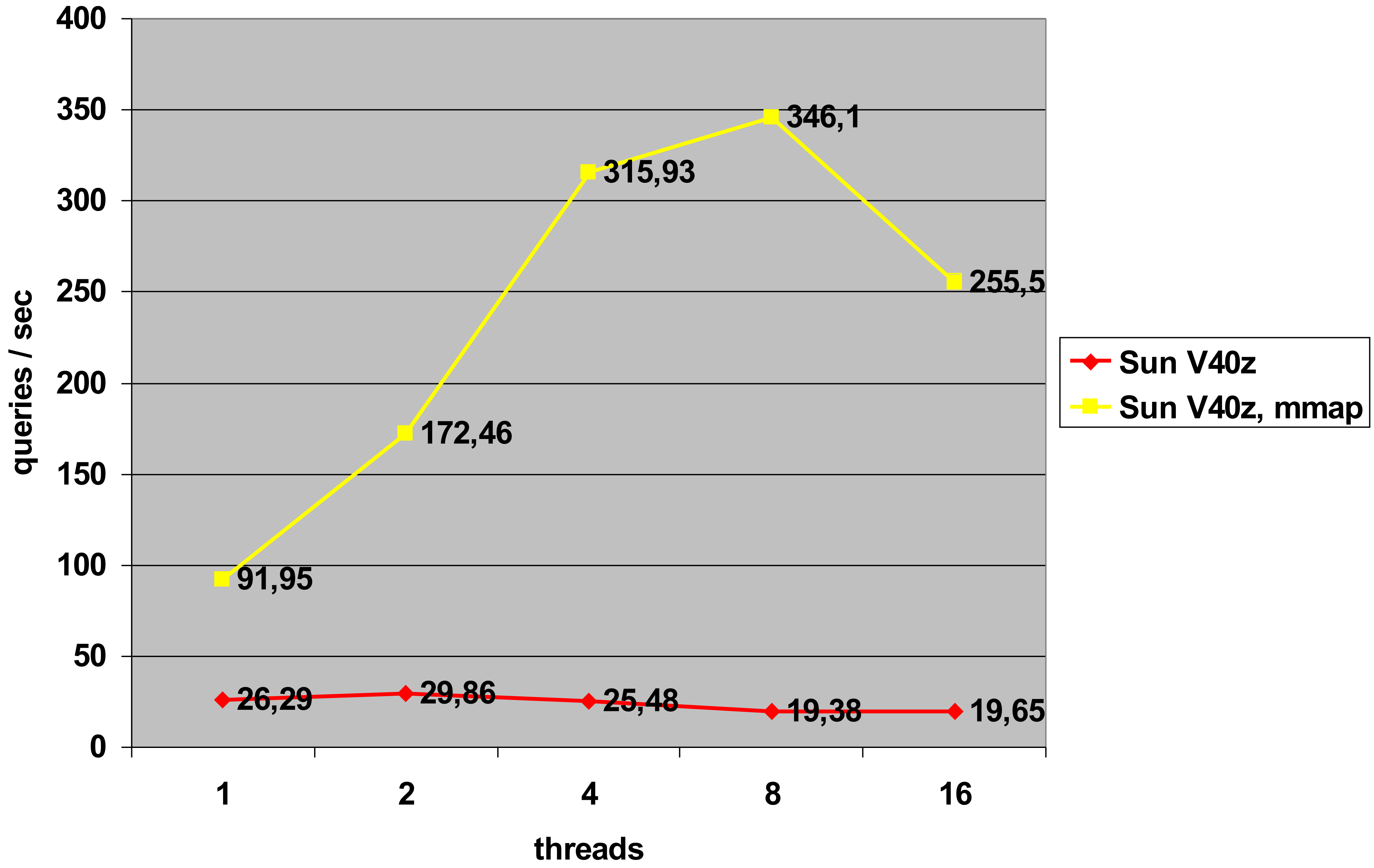
pread64 is called for each row / 20000 times per query

What to do?

Main idea is: avoid pread calls

- memory mapping functions
  - Mmap / memcpy
- Implemented in 5.1
  - --myisam_use_mmap

Results, --myisam_use_mmap



Mmap tricks

- Insert extends file
  - Re-mmap must be called
  - Remap requires exclusive access to file
  - Currently
    - Insert uses pwrite call
    - Remap is postponed to exclusive operation (updated / delete / insert inside file)
  - No performance gain on insert operations
- ~4GB file limit on 32bit systems
  - sliding window can be used

InnoDB

```
CREATE TABLE `b`
( `child_id` int(10) unsigned NOT NULL default '0',
`b` char(20) default NULL,
KEY `child_id` (`child_id`) )
ENGINE=InnoDB
```

Query: `SELECT sql_calc_found_rows * FROM b LIMIT 5;`

- Full scan query, table size is 1 mil rows
- All data is in buffer_pool

Quadxeon (identical problem on Opteron CPU)

- 1 client – 12 sec
- Each of 4 clients – 76 sec

Do you have any idea why?

Profiling

Vmstat

```
1 client                   4 clients
cs  us  sy  id  wa    |     cs  us  sy  id  wa
 55 13   0  87   0    |    442  50   0  50   0
 44 13   0  88   0    |    484  50   0  50   0
 58 13   0  88   0    |    265  50   0  50   0
```

CPU bound problem, how to profile it on Linux?

- Gprof
- Oprofile
- Intel Vtune (commercial)

Perfprof

http://sf.net/projects/perfprof

- True callgraph
- No recompile needed
- Locking / wait profiling

Where does InnoDB spend all this time?

mutex lock

operations with buffer pool

```
~48% of total CPU time
pthread_mutex_trylock()
mutex_spin_wait [sync0sync.c]

buf_page_optimistic_get_func [os0sync.ic]

btr_pcur_restore_position [btr0pcur.c]

sel_restore_position_for_mysql [row0sel.c]

row_search_for_mysql [row0sel.c]

ha_innobase::general_fetch(char*, unsigned, unsigned)
    [ha_innodb.cc]
ha_innobase::rnd_next(char*) [ha_innodb.cc]

rr_sequential [records.cc]


~47% of total CPU time
pthread_mutex_trylock()
mutex_spin_wait [sync0sync.c]

buf_page_release [sync0sync.ic]

mtr_memo_slot_release [mtr0mtr.c]

mtr_commit [mtr0mtr.c]

row_search_for_mysql [row0sel.c]
```

What does it mean?

InnoDB uses its own mutexes:

```
mutex_spin_wait()
{

  for (i=0; i< innodb_sync_spin_loops; i++){
    pthread_mutex_trylock()

  }

  if still not locked
    pthread_cond_wait()

}
```

Mutex in `buf_page_optimistic_get_func / buf_page_release`, buffer_pool mutex is called too often

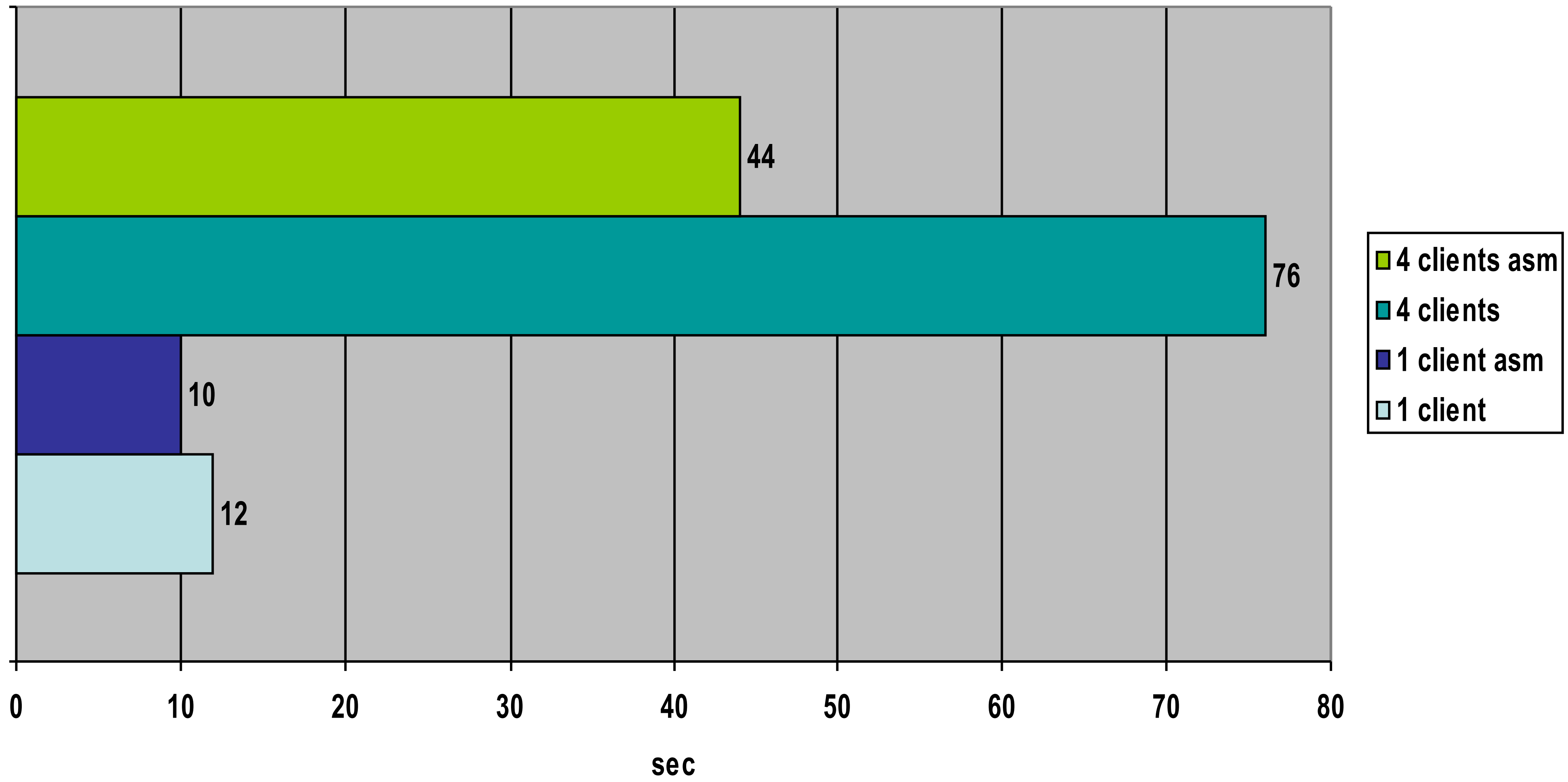
For each row / 1 000 000 times per query

What to do?

- Buffer_pool mutex should be called rarely
  - InnoDB team works on solution
- CPU atomic instructions instead of pthread_mutex_trylock()

Asm locks improve things, but…

Only way: buffer_pool mutex should not block each row, especially in SELECT only workload



Synchronization primitives

POSIX, pthread_mutex_lock

Spin of pthread_mutex_trylock, then …mutex_lock if not succeed (MySQL 5.1 / Linux)

Compatible with pthread_cond_vars

```
for(i= 0; i < SPIN_COUNT; i++)
{
  res= pthread_mutex_trylock(mut);
  if (res == 0)
    return 0;
  if (res != EBUSY)
    return res;
}
return pthread_mutex_lock(mut);
```

Cont.

CPU TEST_AND_SET

```
while (test_and_set(lock))
   while(*((volatile lock_t *) (lock)))
   {
      if (loops++ > SPIN_COUNT)
      {
        pthread_yield();
        loops=0;
      }
   }
```

Main question: how big should be SPIN_LOOP

Alternatives – Anderson's lock and CHAIN based algorithms

Too complex, give benefits on 16+ CPU boxes

Let’s test it

Test program

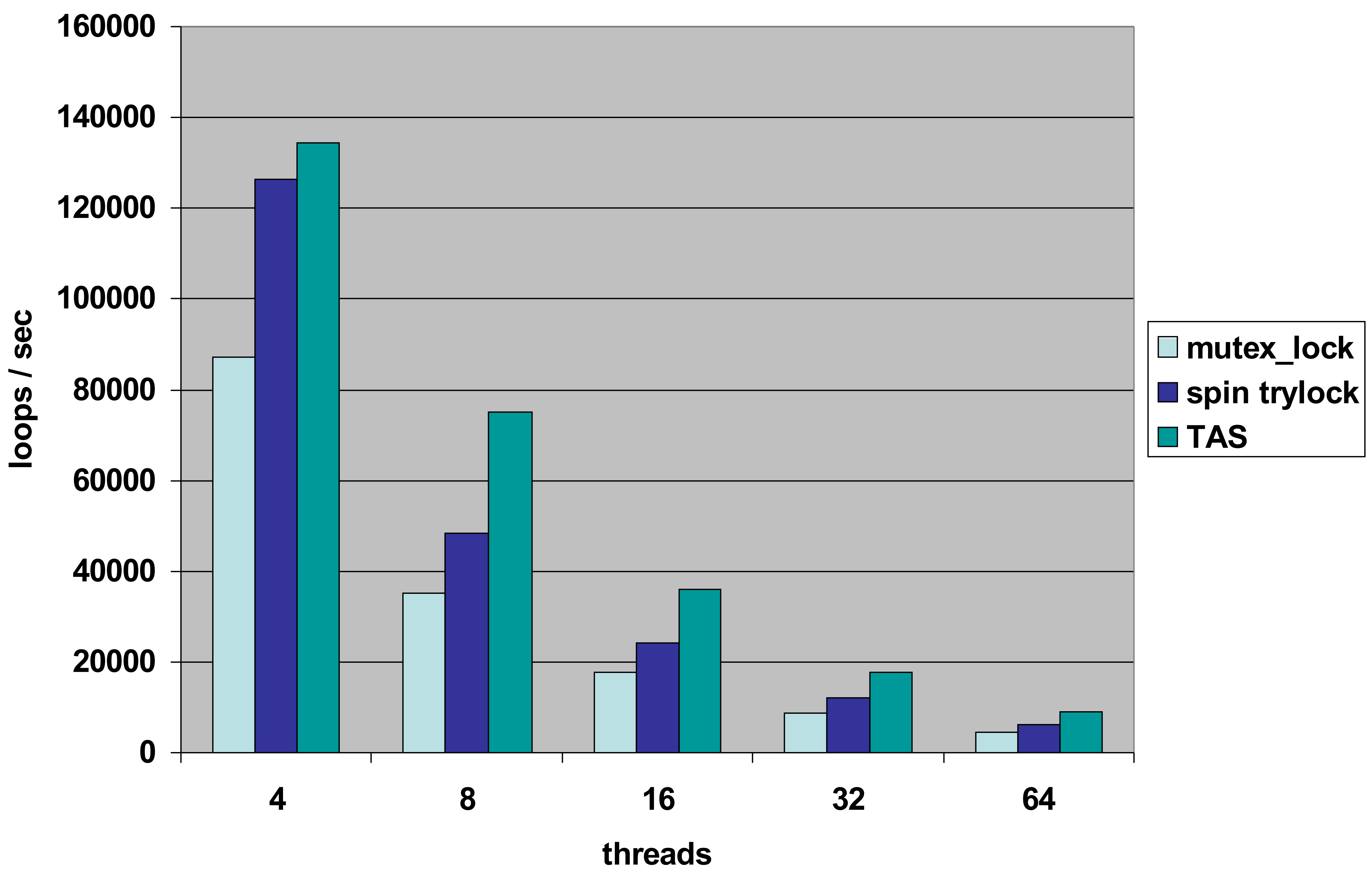
Run loops with mutex protected (critical) and unprotected section

2. Protected section is smaller by 5 times than unprotected
3. Protected section == unprotected
4. Protected section is bigger by 5 times than unprotected

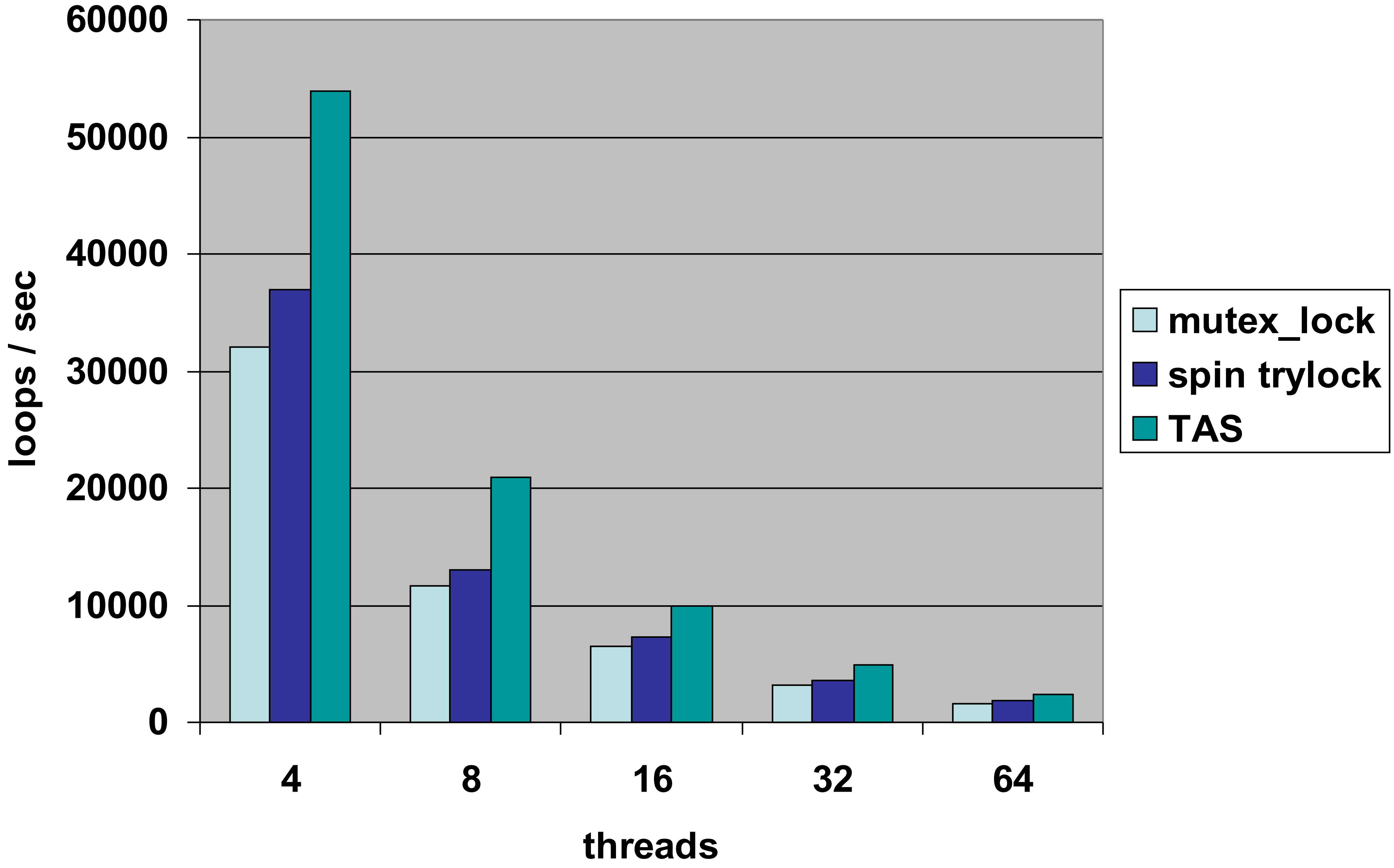
4, 8,16, 32, 64 threads

SPIN_COUNT = 20

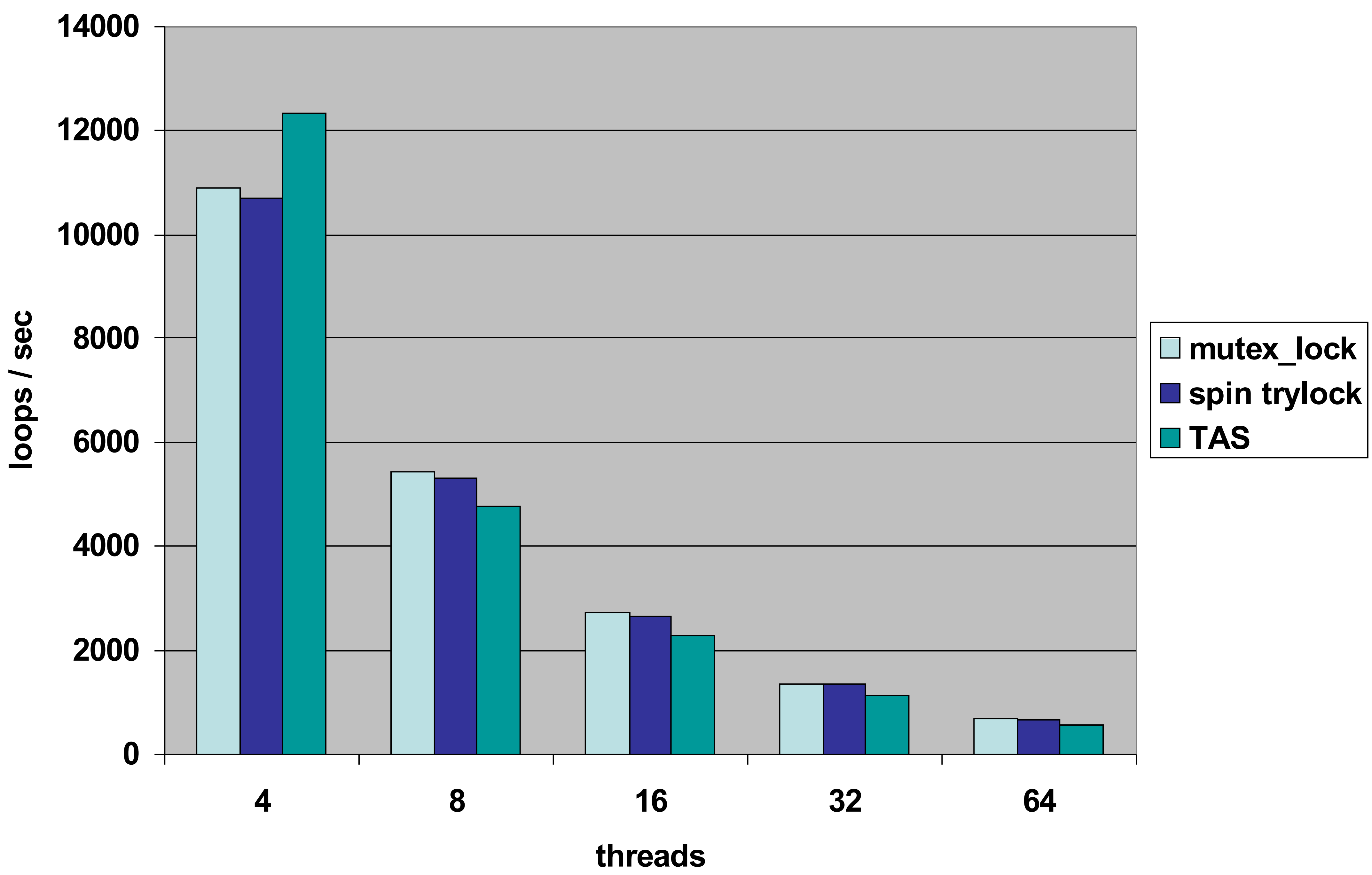
Small critical section



Critical == unprotected



Critical is bigger



Conclusion

- No ideal solution
- SPIN_COUNT should be adaptive in depend of critical section length
  - Complex algorithm, task for investigations

Final

Synchronization primitives

- Don't overuse it
  - Developers does not design the concurrency in details
- Mutexes are designed to protect only several instruction
- Think about a non-standard implementation

System calls

- Can be more expensive than you expected

Thank you!

Questions?

Write us vadim@mysql.com, peter@mysql.com