

sysbench 1.0: teaching an old dog new tricks

Alexey Kopytov

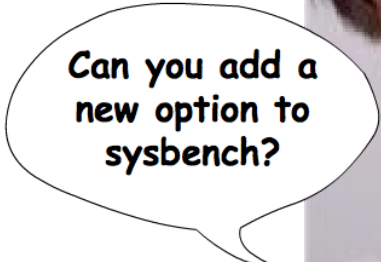
akopytov@gmail.com

The early days (2004)

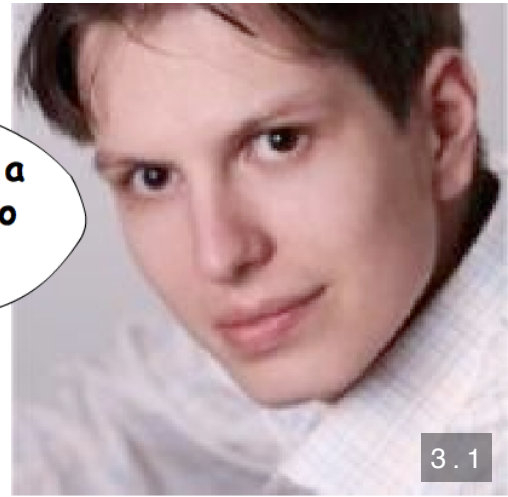
- started as an internal project in High Performance Group @ MySQL AB
- the very first version written by Peter Zaitsev
- I took over shortly after joining the team
- contained SQL ("OLTP"), file, memory, cpu and scheduler benchmarks
- proved to be very useful in identifying performance problems, troubleshooting customer issues, etc.

Growing complexity (2005-2006)

- lots of internal feature requests (mostly from Peter)
- non-trivial inter-dependencies
- impossible to cover all possible use cases
- code unmaintainable by 2006



Can you add a new option to sysbench?



Let's make it scriptable!

- let users define workloads with a high-level API
- let sysbench do all the heavy lifting: threads, statistics, random numbers, DB abstraction
- OLTP benchmarks rewritten as Lua scripts in sysbench 0.5



Why Lua?

- the "speed queen" of dynamic languages
- designed to be embedded into C/C++ applications
- simple and elegant, but powerful
- Lua in 15 minutes:

<https://learnxinyminutes.com/docs/lua/>

SQL benchmarks in Lua

- predefined hooks called from C code
- API for SQL and random numbers/strings generation written in C and used from Lua code

```
function prepare()  
  db_query("CREATE TABLE t (a INT)")  
  db_query("INSERT INTO t VALUES (1)")  
end  
function event()  
  db_query("UPDATE t SET a = a + " .. sb_rand(1, 1000))  
end  
function cleanup()  
  db_query("DROP TABLE t")  
end
```

```
$ sysbench --test=test.lua prepare # calls prepare()
```

```
$ sysbench --test=test.lua --num-threads=16 --report-interval=1 run # calls event() in a loop  
[ 1s] threads: 16, tps: 0.00, reads: 0.00, writes: 13788.65, response time: 1.43ms (95%)  
[ 2s] threads: 16, tps: 0.00, reads: 0.00, writes: 14067.56, response time: 1.40ms (95%)  
...
```

Development hiatus (2007-2015)

- sysbench worked well for a wide range of use cases
- used by many individuals, companies to benchmark MySQL or for internal QA
- stopped active development after moving to MySQL Development (and then Percona)
- reports about scalability issues on high-end hardware starting from 2012

Restarted development (2016+)

- started working with sysbench again
- a major refactoring effort to address performance issues and functional limitations
- announced the start of the project in my blog, but failed to report progress
- however...



***BREAKING
NEWS***

Announcing sysbench 1.0:

- the first release since 0.4.12 (~2006!)
- closes issue #1 "Release of sysbench"
- much better performance and scalability
- improved command line syntax
- extended SQL API
- latency histograms
- error hooks
- report hooks
- custom and parallel commands



Performance improvements

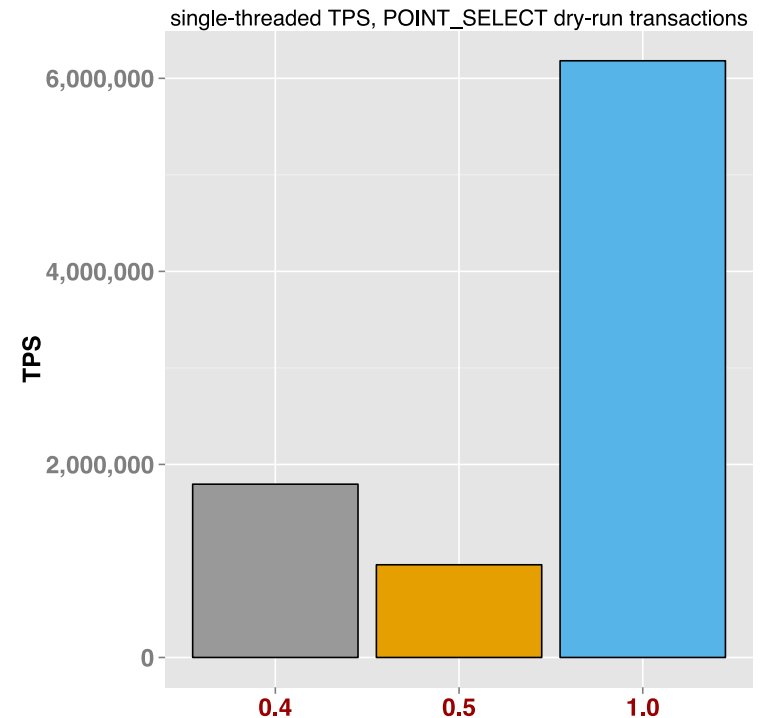
How to benchmark a benchmark utility?

- `sysbench --mysql-dry-run`

Single-threaded performance

Optimizations in 1.0:

- LuaJIT:
 - faster Lua code execution
 - faster C calls with FFI
- optimized event loop
- faster PRNG (xoroshiro128+)
- 3.44x faster than 0.4
- 6.44x faster than 0.5

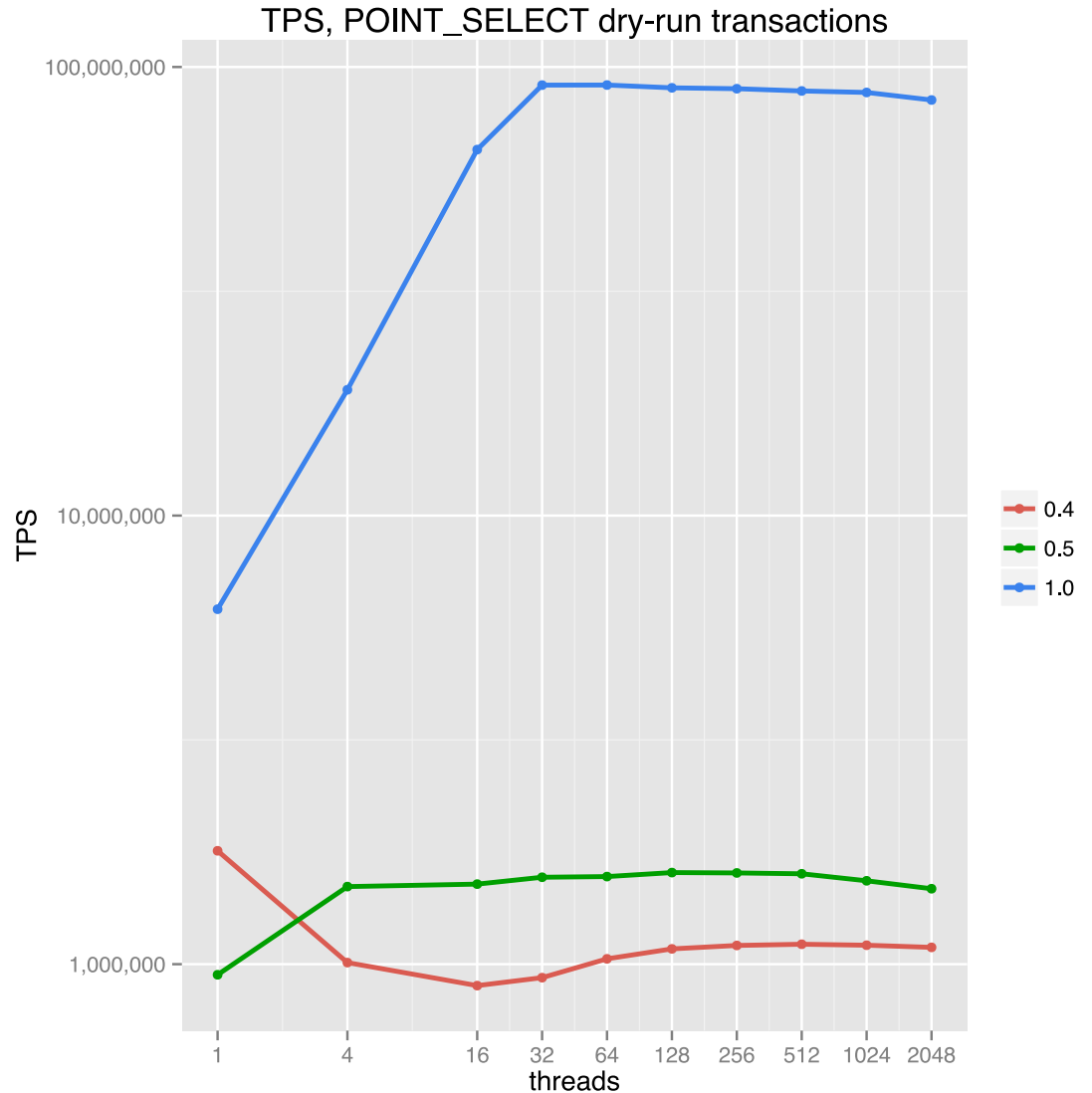


Scalability

threads	0.4	0.5	1.0
1	1789514	947123	6184301
4	1008154	1489174	19073059
16	895810	1508292	65444876
32	933098	1562345	91118515
64	1027856	1567786	91157330
128	1081680	1600286	89853314
256	1100908	1597260	89449255
512	1107764	1590471	88422934
1024	1102249	1534225	87745092
2048	1090127	1473032	84412932

Changes in 1.0:

- ConcurrencyKit
- no mutexes
- no shared counters



Command line syntax change

- sysbench 0.5:

```
$ sysbench --test=<path> [options...] command
```

- sysbench 1.0:

```
$ sysbench [<path>] [options...] [command]
```

or even:

```
#!/usr/bin/env sysbench
function event()
    db_query("SELECT 1")
end
```

```
$ chmod +x mybench.lua
```

```
$ ./mybench.lua run
```

```
[ 1s ] thds: 1 tps: 15295.05 qps: 15295.05 (r/w/o: 15295.05/0.00/0.00) lat (ms,95%): 0.09 err/s: 0.00
[ 2s ] thds: 1 tps: 21934.19 qps: 21934.19 (r/w/o: 21934.19/0.00/0.00) lat (ms,95%): 0.06 err/s: 0.00
[ 3s ] thds: 1 tps: 22785.35 qps: 22785.35 (r/w/o: 22785.35/0.00/0.00) lat (ms,95%): 0.06 err/s: 0.00
```

```
^C
```

Command line options

- problem with option validation in sysbench 0.5:
 - no way for Lua scripts to declare supported options
 - **all** command line options are exported to Lua as global variables

```
$ sysbench oltp.lua --oltp-tbaales-count=8 run # no error, assumes --oltp-tables-count=1
```

- default values were handled manually:

```
oltp_table_size = oltp_table_size or 10000
if (oltp_create_secondary == 'off') then
  oltp_create_secondary = false
else
  oltp_create_secondary = true
end
```

Command line options

- sysbench 1.0:
 - scripts can *declare* their options, so sysbench can *validate* them

```
sysbench.cmdline.options = {  
  tables = {"Number of tables", 1},  
  table_size = {"Number of rows per table", 10000},  
  create_secondary = {"Create a secondary key", true}  
}
```

```
$ sysbench --tbales=8 mybench.lua run  
invalid option: --tbales=8  
  
$ sysbench mybench.lua help  
mybench.lua options:  
  --table_size=N Number of rows per table [10000]  
  --tables=N      Number of tables [1]
```

- bundled OLTP Lua scripts declare their options, respond to `help` command

Using C library with LuaJIT

- plain Lua (sysbench 0.5):

```
function event()  
  db_query("SELECT 1")  
  os.execute("sleep 1") -- ugly!  
  db_query("SELECT 2")  
end
```

- LuaJIT + Foreign Functions Interface (sysbench 1.0)
 - *allows calling external C functions and using C data structures from pure Lua code*

```
ffi = require("ffi")  
ffi.cdef("int usleep(int microseconds);")  
  
function event()  
  db_query("SELECT 1")  
  ffi.C.usleep(1000)  
  db_query("SELECT 2")  
end
```

New SQL API

```
function thread_init()  
    drv = sysbench.sql.driver()  
    con = drv:connect()  
end  
  
function event()  
    con:query("SELECT 1")  
end  
  
function thread_done()  
    con:disconnect()  
end
```

- use LuaJIT FFI for better performance
- bundled OLTP scripts rewritten to the new API

New SQL API: multiple connections per thread

- sysbench 0.5:

```
db_query("SELECT 1") -- works with a single automatically created connection
```

- sysbench 1.0:

```
c1 = drv:connect() -- create as many connections  
c2 = drv:connect() -- as you like  
  
c1:query("SELECT 1")  
c2:query("SELECT 2")
```

New SQL API: results sets

- sysbench 0.5 discarded all results automatically
- processing results is required by some complex benchmark scenarios (e.g. LinkBench)
- sysbench 1.0:

```
c1 = sysbench.sql.driver():connect()
c1:query("CREATE TABLE t (a INT, b VARCHAR(255))")
c1:query([[INSERT INTO t VALUES (1, "foo"), (2, "bar")]])

rs = c1:query("SELECT * FROM t")
for i = 1, rs.nrows do
    row = rs:fetch_row()
    print(row[1], row[2])
end

print(c1:query_row("SHOW GLOBAL STATUS LIKE 'Handler_read_rnd_next'))
```

```
$ sysbench test.lua
1      foo
2      bar
Handler_read_rnd_next  11718125
```

Latency histograms

```
ffi.cdef("int usleep(int microseconds);")
```

```
function event()  
  ffi.C.usleep(1000)  
end
```

```
$ sysbench test.lua --events=100 --histogram run
```

```
Latency histogram (values are in milliseconds)
```

value	----- distribution -----	count
1.044	**	2
1.063	*****	28
1.082	*****	33
1.102	*****	22
1.122	*****	13
1.142	**	2

```
General statistics:
```

total time:	0.1119s
total number of events:	100

```
Latency (ms):
```

min:	1.06
avg:	1.09
max:	1.16
95th percentile:	1.10
sum:	109.23

Error hooks

- problem: special handling for specific SQL errors
 - restart transactions on deadlocks
 - reconnect to out-of-sync cluster node
 - route queries to another node
- solution in sysbench 0.5:
 - `--mysql-ignore-errors=1213,1020`

Error hooks

- solution in sysbench 1.0:
 - reconnect to same node on `ER_UNKNOWN_COM_ERROR`

```
function sysbench.hooks.sql_error_ignorable(err)
  if err.sql_errno == 1047 then -- ER_UNKNOWN_COM_ERROR
    print("Node is out of sync, waiting to reconnect...")
    con:reconnect()
    return true
  end
end
```

- reconnect to a new node on `CR_SERVER_LOST`

```
function sysbench.hooks.sql_error_ignorable(err)
  if err.sql_errno == 2013 then -- CR_SERVER_LOST
    print("Node is down, reconnecting to a new one...")
    con = drv:connect()
    return true
  end
end
```

Custom commands

- sysbench 0.4 / 0.5:
 - predefined set: `prepare`, `run`, `cleanup`, `help`
- sysbench 1.0:
 - scripts can define their own commands:

```
sysbench.cmdline.commands = {  
  prewarm = {cmd_prewarm}  
}  
  
function cmd_prewarm()  
  print("Loading sbtest1 into buffer pool...")  
  db_query("SELECT AVG(id) FROM sbtest1 FORCE KEY (PRIMARY)")  
  db_query("SELECT COUNT(*) FROM sbtest1 WHERE k LIKE '%0%'")  
end
```

```
$ sysbench mybench.lua prewarm
```

```
Loading sbtest1 into buffer pool...
```


Parallel commands

- sysbench 0.4 / 0.5: all commands except `run` executed in a single thread
- sysbench 1.0:
 - can declare custom commands supporting parallel execution:

```
sysbench.cmdline.commands = {  
  prepare = {parallel_prepare, sysbench.cmdline.PARALLEL_COMMAND}  
}  
  
function parallel_prepare()  
  db_query("CREATE TABLE sbtest" .. sysbench.tid .. "(a INT)");  
  db_query("INSERT INTO sbtest" .. sysbench.tid .. " VALUES (...)"  
end
```

Custom reports

- standard human-readable reports in sysbench:

```
[ 8s ] thds: 32 tps: 11580.79 qps: 232597.61 (r/w/o: 162993.88/46390.16/23213.57) lat (ms,95%): 4.10 err/s: 52.0
[ 9s ] thds: 32 tps: 11703.11 qps: 234551.37 (r/w/o: 164282.69/46826.45/23442.23) lat (ms,95%): 3.96 err/s: 35.0
SQL statistics:
  queries performed:
    read:                1678180
    write:               478000
    other:               239239
    total:               2395419
  transactions:         119369 (11926.57 per sec.)
  queries:              2395419 (239334.51 per sec.)
  ignored errors:       501 (50.06 per sec.)
  reconnects:           0 (0.00 per sec.)

General statistics:
  total time:           10.0069s
  total number of events: 119369

Latency (ms):
  min:                 1.42
  avg:                 2.68
  max:                 15.78
  95th percentile:    4.10
  sum:                 319811.19
```

- hard to parse into a machine-readable format

Long-requested feature

- sysbench 1.0: hooks to print statistics in custom formats
- example: CSV

```
function sysbench.hooks.report_intermediate(stat)
  local seconds = stat.time_interval
  print(string.format("%.0f,%u,%4.2f,%4.2f,%4.2f,%4.2f,%4.2f,%4.2f,%4.2f",
    stat.time_total, stat.threads_running,
    stat.events / seconds, (stat.reads + stat.writes + stat.other) / seconds,
    stat.reads / seconds, stat.writes / seconds, stat.other / seconds,
    stat.latency_pct * 1000, stat.errors / seconds, stat.reconnects / seconds))
end
```

```
$ sysbench test.lua --threads=32 --report-interval=1 run
1,32,12227.49,245589.45,172087.82,48972.82,24528.81,3.89,43.90,0.00
2,32,12580.84,252341.05,176742.96,50390.39,25207.70,3.68,44.01,0.00
3,32,12594.35,252761.04,177069.93,50451.40,25239.70,3.55,54.00,0.00
4,32,12377.77,248495.40,174108.78,49571.08,24815.54,4.03,57.00,0.00
5,32,12495.12,250733.49,175668.75,50026.50,25038.25,3.75,48.00,0.00
6,32,12451.92,249896.37,175062.86,49875.67,24957.84,3.96,53.00,0.00
7,32,12208.90,244758.96,171428.57,48874.59,24455.80,4.25,40.00,0.00
8,32,12109.62,243071.29,170291.57,48508.48,24271.24,4.25,50.99,0.00
9,32,12335.24,247441.91,173355.47,49365.96,24720.49,4.10,50.01,0.00
```

Custom reports: JSON example

```
function sysbench.hooks.report_intermediate(stat)
  local seconds = stat.time_interval
  print(string.format([[
{
  "time": %4.0f,
  ...
}], stat.time_total, stat.threads_running, stat.events / seconds,
  (stat.reads + stat.writes + stat.other) / seconds, stat.reads / seconds,
  stat.writes / seconds, stat.other / seconds, stat.latency_pct * 1000,
  stat.errors / seconds, stat.reconnects / seconds))
end
```

```
$ sysbench test.lua --threads=32 --report-interval=1 run
{
  "time": 7,
  "threads": 32,
  "tps": 12003.44,
  "qps": {
    "total": 240990.88,
    "reads": 168816.22,
    "writes": 48114.77,
    "other": 24059.89,
  },
  "latency": 4.33,
  "errors": 52.00,
  "reconnects": 0.00
},
```

Custom reports

- store results in Prometheus/Graphite/etc.
- get custom perf. metrics from OS or MySQL server:

```
sysbench.hooks.report_intermediate =  
  function (stat)  
    if con == nil then  
      con = assert(sysbench.sql.driver():connect())  
    end  
    sysbench.report_default(stat)  
    name, avglat = con:query_row([[  
SELECT event_name AS event, avg_timer_wait as avg_latency  
  FROM performance_schema.events_waits_summary_global_by_event_name  
 WHERE event_name != 'idle'  
   AND sum_timer_wait > 0  
 ORDER BY sum_timer_wait DESC LIMIT 1;]])  
    print("top wait event: "..name.." avg_latency: "..avglat)  
  end
```

```
[ 1s ] thds: 1 tps: 492.84 qps: 9869.74 (r/w/o: 6911.71/1971.36/986.68) lat (ms,95%): 2.35 err/s 0.00 reconn/s  
top wait event: wait/io/file/innodb/innodb_data_file avg_latency: 176826163
```

Legacy Lua API

What about old scripts?

- "old" sysbench has been around long enough
- they will still work at least until the next major release
- there are regression tests to verify legacy API is functional

Help wanted!

- Unsupported drivers:
 - Oracle RDBMS
 - Drizzle
 - libattachsql
- PostgreSQL driver:
 - supported, but needs more work



Windows support

Supporting Windows:



- incomplete C99 support in MSVC
- no support in ConcurrencyKit
- but patches are welcome

The Future:

- documentation
- packaging
- syslinkbench
- MongoDB driver
- MySQL X Protocol driver

Summing-up

- sysbench 1.0 is the most significant milestone so far
- hope it will be as useful for you as it is for me
- <https://github.com/akopytov/sysbench>
- these slides: <http://kaamos.me/talks/fosdem17>

Thank you! Questions?