SysBench manual

Alexey Kopytov

〈kaamos@users.sourceforge.net〉

Table of Contents

Chapter 1. Introduction

SysBench is a modular, cross-platform and multi-threaded benchmark tool for evaluating OS parameters that are important for a system running a database under intensive load.

The idea of this benchmark suite is to quickly get an impression about system performance without setting up complex database benchmarks or even without installing a database at all.

1. Features of SysBench

Current features allow to test the following system parameters:

- file I/O performance
- scheduler performance
- memory allocation and transfer speed
- POSIX threads implementation performance
- database server performance

2. Design

The design is very simple. SysBench runs a specified number of threads and they all execute requests in parallel. The actual workload produced by requests depends on the specified test mode. You can limit either the total number of requests or the total time for the benchmark, or both.

Available test modes are implemented by compiled-in modules, and SysBench was designed to make adding new test modes an easy task. Each test mode may have additional (or workload-specific) options.

3. Links

Home page

http://sysbench.sf.net/.

Download

http://sf.net/projects/sysbench/.

Mailing lists

sysbench-general

Web forums

- Developers
- Help
- Open discussion

Bug tracking system

- Bug reports
- Feature requests

4. Installation

If you are building SysBench from a Bazaar repository rather than from a release tarball, you should run **./autogen.sh** before building.

The following standart procedure will be sufficient to build SysBench in most cases:

```
./configure
make
make install
```

The above procedure will try to compile SysBench with MySQL support by default. If you have MySQL headers and libraries in non-standard locations (and no**mysql_config** can be found in the `PATH` environmental variable), then you can specify them explicitly with `--with-mysql-includes` and `--with-mysql-libs` options to**./configure**.

To compile SysBench without MySQL support, use `--without-mysql`. In this case all database-related test modes will be unavailable.

If you are running on a 64-bit platform, make sure to build a 64-bit binary by passing the proper target platform and compiler options to **configure** script. You can also consult the INSTALL file for generic installation instructions.

Chapter 2. Usage

1. General syntax

The general syntax for SysBench is as follows:

```
sysbench [common-options] --test=name [test-options] command
```

See Section 2, “Common command line options” for a description of common options and documentation for particular test mode for a list of test-specific options.

Below is a brief description of available commands and their purpose:

prepare
: Performs preparative actions for those tests which need them, e.g. creating the necessary files on disk for the `fileio` test, or filling the test database for the `oltp` test.

run
: Runs the actual test specified with the `--test=`*name* option.

cleanup
: Removes temporary data after the test run in those tests which create one.

help
: Displays usage information for a test specified with the `--test=`*name* option.

Also you can use **sysbench help** to display the brief usage summary and the list of available test modes.

2. Common command line options

The table below lists the supported common options, their descriptions and default values:

Option	*Description*	*Default value*

--num-threads	The total number of worker threads to create	1
--max-requests	Limit for total number of requests. 0 means unlimited	10000
--max-time	Limit for total execution time in seconds. 0 (default) means unlimited	0
--forced-shutdown	Amount of time to wait after --max-time before forcing shutdown. The value can be either an absolute number of seconds or as a percentage of the --max-time value by specifying a number of percents followed by the '%' sign. "off" (the default value) means that no forced shutdown will be performed.	off
--thread-stack-size	Size of stack for each thread	32K
--init-rng	Specifies if random numbers generator should be initialized from timer before the test start	off
--test	Name of the test mode to run	*Required*
--debug	Print more debug info	off
--validate	Perform validation of test results where possible	off
--help	Print help on general syntax or on a test mode specified with --test, and exit	off
--verbosity	Verbosity level (0 - only critical messages, 5 - debug)	4
--percentile	SysBench measures execution times for all processed requests to display statistical information like minimal, average and maximum execution time. For most benchmarks it is also useful to know a request execution time value matching some percentile (e.g. 95% percentile means we should drop 5% of the most long requests and choose the maximal value from the remaining ones). This option allows to specify a percentile rank of query execution times to count	95
--batch	Dump current results periodically (see Section 3, "Batch mode")	off

--batch-delay	Delay between batch dumps in secods (see Section 3, “Batch mode”)	300
--validate	Perform validation of test results where possible	off

Note that numerical values for all *size* options (like `--thread-stack-size` in this table) may be specified by appending the corresponding multiplicative suffix (K for kilobytes, M for megabytes, G for gigabytes and T for terabytes).

3. Batch mode

In some cases it is useful to have not only the final benchmarks statistics, but also periodical dumps of current stats to see how they change over the test run. For this purpose SysBench has a batch execution mode which is turned on by the `--batch` option. You may specify the delay in seconds between the consequent dumps with the `--batch-delay` option. Example:

```
sysbench --batch --batch-delay=5 --test=threads run
```

This will run SysBench in a threads test mode, with the current values of minimum, average, maximum and percentile for request execution times printed every 5 seconds.

4. Test modes

This section gives a detailed description for each test mode available in SysBench.

4.1. cpu

The `cpu` is one of the most simple benchmarks in SysBench. In this mode each request consists in calculation of prime numbers up to a value specified by the `--cpu-max-primes` option. All calculations are performed using 64-bit integers.

Each thread executes the requests concurrently until either the total number of requests or the total execution time exceed the limits specified with the common command line options.

Example:

```
sysbench --test=cpu --cpu-max-prime=20000 run
```

4.2. threads

This test mode was written to benchmark scheduler performance, more specifically the cases when a scheduler has a large number of threads competing for some set of mutexes.

SysBench creates a specified number of threads and a specified number of mutexes. Then each thread starts running the requests consisting of locking the mutex, yielding the CPU, so the thread is placed in the run queue by the scheduler, then unlocking the mutex when the thread is rescheduled back to execution. For each request, the above actions are run several times in a loop, so the more iterations is performed, the more concurrency is placed on each mutex.

The following options are available in this test mode:

Option	*Description*	*Default value*
--thread-yields	Number of *lock/yield/unlock* loops to execute per each request	1000
--thread-locks	Number of mutexes to create	8

Example:

```
sysbench --num-threads=64 --test=threads --thread-yields=100
--thread-locks=2 run
```

4.3. mutex

This test mode was written to emulate a situation when all threads run concurrently most of the time, acquiring the mutex lock only for a short period of time (incrementing a global variable). So the purpose of this benchmarks is to examine the performance of mutex implementation.

The following options are available in this test mode:

Option	*Description*	*Default value*
--mutex-num	Number of mutexes. The actual mutex to lock is chosen randomly before each lock	4096
--mutex-locks	Number of mutex locks to acquire per each request	50000
--mutex-loops	Number of iterations for an empty loop to perform before acquiring the lock	10000

4.4. memory

This test mode can be used to benchmark sequential memory reads or writes. Depending on command line options each thread can access either a global or a local block for all memory operations.

The following options are available in this test mode:

Option	*Description*	*Default value*
--memory-block-size	Size of memory block to use	1K
--memory-scope	Possible values: global, local. Specifies whether each thread will use a globally allocated memory block, or a local one.	global
--memory-total-size	Total size of data to transfer	100G
--memory-oper	Type of memory operations. Possible values: read, write.	100G

4.5. fileio

This test mode can be used to produce various kinds of file I/O workloads. At the prepare stage SysBench creates a specified number of files with a specified total size, then at the run stage, each thread performs specified I/O operations on this set of files.

When the global `--validate` option is used with the `fileio` test mode,

SysBench performs checksums validation on all data read from the disk. On each write operation the block is filled with random values, then the checksum is calculated and stored in the block along with the offset of this block within a file. On each read operation the block is validated by comparing the stored offset with the real offset, and the stored checksum with the real calculated checksum.

The following I/O operations are supported:

seqwr
: sequential write

seqrewr
: sequential rewrite

seqrd
: sequential read

rndrd
: random read

rndwr
: random write

rndrw
: combined random read/write

Also, the following file access modes can be specified, if the underlying platform supports them:

Asynchronous I/O mode
: At the moment only Linux AIO implementation is supported. When running in asynchronous mode, SysBench queues a specified number of I/O requests using Linux AIO API, then waits for at least one of submitted requests to complete. After that a new series of I/O requests is submitted.

Slow `mmap()` mode

: In this mode SysBench will use `mmap`'ed I/O. However, a

: separate `mmap` will be used for each I/O request due to the limitation of

: 32-bit architectures (we cannot`mmap()` the whole file, as its size migth

: possibly exceed the maximum of 2 GB of the process address space).

Fast `mmap()` mode

On 64-bit architectures it is possible to mmap() the whole file into the process address space, avoiding the limitation of 2 GB on 32-bit platforms.

Using fdatasync() instead of fsync()

Additional flags to open(2)

SysBench can use additional flags to open(2), such as O_SYNC, O_DSYNC and O_DIRECT.

Below is a list of test-specific option for the **fileio** mode:

Option	*Description*	*Default value*
--file-num	Number of files to create	128
--file-block-size	Block size to use in all I/O operations	16K
--file-total-size	Total size of files	2G
--file-test-mode	Type of workload to produce. Possible values: seqwr, seqrewr, seqrd, rndrd, rndwr, rndwr (see above)	*required*
--file-io-mode	I/O mode. Possible values: sync, async, fastmmap, slowmmap (only if supported by the platform, see above).	sync
--file-async-backlog	Number of asynchronous operations to queue per thread (only for --file-io-mode=async, see above)	128
--file-extra-flags	Additional flags to use with open(2)	
--file-fsync-freq	Do fsync() after this number of requests (0 - don't use fsync())	100
--file-fsync-all	Do fsync() after each write operation	no
--file-fsync-end	Do fsync() at the end of the test	yes
--file-fsync-mode	Which method to use for synchronization. Possible values: fsync, fdatasync (see above)	fsync
--file-merge	Merge at most this number of I/O requests if possible	0

d-requests	(0 - don't merge)	
--file-rw-ratio	reads/writes ration for combined random read/write test	1.5

Usage example:

```
$ sysbench --num-threads=16 --test=fileio --file-total-size=3G
--file-test-mode=rndrw prepare
$ sysbench --num-threads=16 --test=fileio --file-total-size=3G
--file-test-mode=rndrw run
$ sysbench --num-threads=16 --test=fileio --file-total-size=3G
--file-test-mode=rndrw cleanup
```

In the above example the first command creates 128 files with the total size of 3 GB in the current directory, the second command runs the actual benchmark and displays the results upon completion, and the third one removes the files used for the test.

4.6. oltp

This test mode was written to benchmark a real database performance. At the **prepare** stage the following table is created in the specified database (`sbtest` by default):

```
CREATE TABLE `sbtest` (
`id` int(10) unsigned NOT NULL auto_increment,
`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`));
```

Then this table is filled with a specified number of rows.

The following execution modes are available at the **run** stage:

Simple

In this mode each thread runs simple queries of the following form:

```
SELECT c FROM sbtest WHERE id=N
```

where *N* takes a random value in range 1..<*table size*>

Advanced transactional

Each thread performs transactions on the test table. If the test table and database support transactions (e.g. InnoDB engine in MySQL), then `BEGIN`/`COMMIT`statements will be used to start/stop a transaction.

Otherwise, SysBench will use `LOCK TABLES`/`UNLOCK TABLES` statements (e.g. for MyISAM engine in MySQL). If some rows are deleted in a transaction, the same rows will be inserted within the same transaction, so this test mode does not destruct any data in the test table and can be run multiple times on the same table.

Depending on the command line options, each transaction may contain the following statements:

- Point queries:

```
SELECT c FROM sbtest WHERE id=N
```

- Range queries:

```
SELECT c FROM sbtest WHERE id BETWEEN N AND M
```

- Range SUM() queries:

```
SELECT SUM(K) FROM sbtest WHERE id BETWEEN N and M
```

- Range ORDER BY queries:

```
SELECT c FROM sbtest WHERE id between N and M ORDER BY c
```

- Range DISTINCT queries:

```
SELECT DISTINCT c FROM sbtest WHERE id BETWEEN N and M ORDER BY
c
```

- UPDATEs on index column:

```
UPDATE sbtest SET k=k+1 WHERE id=N
```

- UPDATEs on non-index column:

```
UPDATE sbtest SET c=N WHERE id=M
```

- DELETE queries:

```
DELETE FROM sbtest WHERE id=N
```

- INSERT queries:

```
INSERT INTO sbtest VALUES (...)
```

Non-transactional

This mode is similar to **Simple**, but you can also choose the query to run. Note that unlike the **Advanced transactional** mode, this one does not preserve the test table between requests, so you should recreate it with the appropriate **cleanup**/**prepare** commands between consecutive benchmarks.

Below is a list of possible queries:

- Point queries:

```
SELECT pad FROM sbtest WHERE id=N
```

- UPDATEs on index column:

```
UPDATE sbtest SET k=k+1 WHERE id=N
```

- UPDATEs on non-index column:

```
UPDATE sbtest SET c=N WHERE id=M
```

- DELETE queries:

```
DELETE FROM sbtest WHERE id=N
```

- The generated row IDs are unique over each test run, so no row is deleted twice.
- INSERT queries:

```
INSERT INTO sbtest (k, c, pad) VALUES(N, M, S)
```

Below is a list of options available for the database test mode:

<table>
<tr><th>Option</th><th>Description</th><th>Default value</th></tr>
<tr><td>--oltp-test-mode</td><td>Execution mode (see above). Possible values: simple (simple), complex (advanced transactional) and nontrx (non-transactional)</td><td>complex</td></tr>
<tr><td>--oltp-read-only</td><td>Read-only mode. No UPDATE, DELETE or INSERT queries will be performed.</td><td>off</td></tr>
<tr><td>--oltp-skip-trx</td><td>Omit BEGIN/COMMIT statements, i.e. run the same queries as the test would normally run but without using transactions.</td><td>off</td></tr>
<tr><td>--oltp-reconnect-mode</td><td>Reconnect mode. Possible values:<table><tr><td>session</td><td>Don't reconnect (i.e. each thread disconnects only at the end of the test)</td></tr><tr><td>query</td><td>Reconnect after each SQL query</td></tr><tr><td>transaction</td><td>Reconnect after each transaction (if transactions are used in the selected DB test)</td></tr><tr><td>random</td><td>One of the above modes is randomly chosen for each transaction</td></tr></table></td><td>session</td></tr>
<tr><td>--oltp-range-size</td><td>Range size for range queries</td><td>100</td></tr>
<tr><td>--oltp-point-selects</td><td>Number of point select queries in a single transaction</td><td>10</td></tr>
<tr><td>--oltp-simple-ranges</td><td>Number of simple range queries in a single transaction</td><td>1</td></tr>
<tr><td>--oltp-sum-ranges</td><td>Number of SUM range queries in a single transaction</td><td>1</td></tr>
<tr><td>--oltp-order-ranges</td><td>Number of ORDER range queries in a single transaction</td><td>1</td></tr>
<tr><td>--oltp-distinct-ranges</td><td>Number of DISTINCT range queries in a single transaction</td><td>1</td></tr>
<tr><td>--oltp-index-updates</td><td>Number of index UPDATE queries in a single transaction</td><td>1</td></tr>
<tr><td>--oltp-non-index-updates</td><td>Number of non-index UPDATE queries in a single transaction</td><td>1</td></tr>
<tr><td>--oltp-nont</td><td>Type of queries for non-transactional execution mode</td><td>select</td></tr>
</table>

rx-mode	(see above). Possible values: select, update_key, update_nokey, insert, delete.	
--oltp-connect-delay	Time in microseconds to sleep after each connection to database	10000
--oltp-user-delay-min	Minimum time in microseconds to sleep after each request	0
--oltp-user-delay-max	Maximum time in microseconds to sleep after each request	0
--oltp-table-name	Name of the test table	sbtest
--oltp-table-size	Number of rows in the test table	10000
--oltp-dist-type	Distribution of random numbers. Possible values: uniform (uniform distribution), gauss (gaussian distribution) and special. With special distribution a specified percent of numbers is generated in a specified percent of cases (see options below).	special
--oltp-dist-pct	Percentage of values to be treated as 'special' (for special distribution)	1
--oltp-dist-res	Percentage of cases when 'special' values are generated (for special distribution)	75
--db-ps-mode	If the database driver supports Prepared Statements API, SysBench will use server-side prepared statements for all queries where possible. Otherwise, client-side (or emulated) prepared statements will be used. This option allows to force using emulation even when PS API is available. Possible values: disable, auto.	auto

Also, each database driver may provide its own options. Currently only MySQL driver is available. Below is a list of MySQL-specific options:

Option	*Description*	*Default value*
--mysql-host	MySQL server host. Starting from version 0.4.5 you may specify a list of hosts separated by commas. In this case SysBench	localhost

	will distribute connections between specified MySQL hosts on a round-robin basis. Note that all connection ports and passwords must be the same on all hosts. Also, databases and tables must be prepared explicitely on each host before executing the benchmark.	
--mysql-port	MySQL server port (in case TCP/IP connection should be used)	3306
--mysql-socket	Unix socket file to communicate with the MySQL server	
--mysql-user	MySQL user	user
--mysql-password	MySQL password	
--mysql-db	MySQL database name. Note SysBench will not automatically create this database. You should create it manually and grant the appropriate privileges to a user which will be used to access the test table.	sbtest
--mysql-table-engine	Type of the test table. Possible values: myisam, innodb, heap, ndbcluster, bdb, maria, falcon, pbxt	innodb
--mysql-ssl	Use SSL connections.	no
--myisam-max-rows	MAX_ROWS option for MyISAM tables (required for big tables)	1000000
--mysql-create-options	Additional options passed to CREATE TABLE.	

Example usage:

```
$ sysbench --test=oltp --mysql-table-engine=myisam
--oltp-table-size=1000000 --mysql-socket=/tmp/mysql.sock
prepare
$ sysbench --num-threads=16 --max-requests=100000 --test=oltp
--oltp-table-size=1000000 --mysql-socket=/tmp/mysql.sock
--oltp-read-only run
```

The first command will create a MyISAM table 'sbtest' in a database 'sbtest' on a MySQL server using `/tmp/mysql.sock` socket, then fill this table with 1M records. The second command will run the actual benchmark with 16 client threads, limiting the total number of request by 100,000.