A Better Way to Benchmark Your Open Source Database
Hello!

Steve Shaw

Open source database @ Intel

Main developer of HammerDB
Introduction
What is HammerDB?

- Not a database!
- Leading open source tool for benchmarking relational databases
- Interfaces
  - Graphical
  - Command Line
  - Web REST interfaces
- Industry standard benchmarks
- High performance and scalability
Open Source

- Hosted by TPC Council since 2019
  - Industry standard body for database benchmarks
- TPC-OSS subcommittee
  - Oversees and approves changes
- v4.1 Released on 22nd April 21
- Source code on GitHub
- Binaries @ GitHub Releases
  - https://www.hammerdb.com/download.html
- Client natively supports Linux and Windows on x64
  - GUI & CLI on both Linux and Windows
- GitHub Release Downloads @
  - https://www.hammerdb.com/stats.html
- Test databases on any platform
Supported Databases

- HammerDB supports the most popular relational databases
- Commercial and open source
- Metrics enable comparison across database engines

<table>
<thead>
<tr>
<th>Rank</th>
<th>DBMS</th>
<th>Database Model</th>
<th>Score Apr 2021</th>
<th>Score Mar 2021</th>
<th>Score Apr 2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Oracle</td>
<td>Relational, Multi-model</td>
<td>1274.92</td>
<td>-46.82</td>
<td>-70.51</td>
</tr>
<tr>
<td>2</td>
<td>MySQL</td>
<td>Relational, Multi-model</td>
<td>1220.69</td>
<td>-34.14</td>
<td>-47.66</td>
</tr>
<tr>
<td>3</td>
<td>Microsoft SQL Server</td>
<td>Relational, Multi-model</td>
<td>1007.97</td>
<td>-7.33</td>
<td>-75.46</td>
</tr>
<tr>
<td>4</td>
<td>PostgreSQL</td>
<td>Relational, Multi-model</td>
<td>553.52</td>
<td>+4.23</td>
<td>+43.66</td>
</tr>
<tr>
<td>5</td>
<td>MongoDB</td>
<td>Document, Multi-model</td>
<td>469.97</td>
<td>+7.58</td>
<td>+31.54</td>
</tr>
<tr>
<td>6</td>
<td>IBM Db2</td>
<td>Relational, Multi-model</td>
<td>157.78</td>
<td>+1.77</td>
<td>-7.85</td>
</tr>
</tbody>
</table>
Supported Workloads

- **TPROC-C = OLTP**
  - Transactional workloads. Row oriented, high read and write throughput.
  - Derived from TPC-C

- **TPROC-H = OLAP**
  - Analytic, Decision Support
  - Focus on ETL
  - high bandwidth reads & minimal writes.
  - Derived from TPC-H

- Using TPCC/TPC-C, TPCH/TPC-H for derived workloads not permitted (trademark violation)
Key Database Benchmarking Concepts

- Parallel benchmarking software
  - Concurrency control must be in database, not in client

- Complex workloads designed to scale and test RDBMS concepts
  - Locking and latching

- Cross reference workloads across multiple database engines
  - Validate concepts

- HammerDB up to 6-7M NOPM on commercial database engines on 2 socket servers
  - High confidence levels that bottlenecks are in database software not HammerDB
HammerDB Programming Languages

- Designed for High Performance and Scalability
- Database commands in SQL
- Application logic in stored procedures
- Database Interfaces in C
- More time in the database, less time in the ‘round trip’
- More system resources for the database, less resources for the benchmark client
  - 3% for HammerDB
  - 20% in sysbench in socket/network layer
Tcl as Glue Language

- Python GIL limits to single-threading
- Tcl as glue language for truly parallel multithreading
- Tcl compiles into bytecode at runtime for high performance
- Co-routines used for event-driven scaling only to prevent bottleneck
- GUI & CLI on same codebase
- Native Tk GUI including 4k UHD scaling
Cached vs Scaled Benchmarks

- Default Mode Cached workload
  - Testing goals
- Cached
  - Less than 5000 warehouses
  - All data in memory
  - 10s to hundreds of sessions
  - CPU & Memory Intensive
  - WAL & Redo Disk Write Intensive
  - Maximum performance at minimal configuration
- Scaled (Event driven scaling)
  - More than 5000 warehouses
  - Event driven scaling
  - Thousands of sessions
  - Middleware needed
  - Larger disk and networking requirement
  - Data Disk Read and Write Intensive
- Perfectly Scaled Configuration = Cached performance
System Configuration
CPU

- Some Linux releases default to CPU powersave mode
  - 33% lower performance

```
sudo vi /etc/default/cpufrequtils
GOVERNOR="performance"
```

```
systemctl restart cpufrequtils
systemctl disable ondemand
```

```
sudo ./cpupower frequency-info
```

analyzing CPU 0:
  driver: intel_pstate

...available cpufreq governors: performance powersave

**current policy: frequency should be within 800 MHz and 3.40 GHz.**

The governor "**performance**" may decide which speed to use within this range.

current CPU frequency: Unable to call hardware

current CPU frequency: 1.03 GHz (asserted by call to kernel)

**boost state support:**
  Supported: yes
  Active: yes
Memory

- For default cached workload
- Size Buffer Pool / Cache large enough to cache the TPROC-C schema
- Use 1GB Huge Pages for PostgreSQL
- MySQL/MariaDB InnoDB
  - `innodb_buffer_pool_size=64000M`
- PostgreSQL
  - `shared_buffers = 64000MB`
  - `huge_pages = on`
I/O WAL and Redo Performance

- Use Highest Performance SSDs for WAL/Redo
  - Intel Optane
  - Low latency writes
- Ensure partitions correctly aligned
- Use 1GB redo log / WAL segment size
- MySQL
  - `innodb_log_file_size=1024M`
  - `innodb_log_files_in_group=32`
- PostgreSQL
  - `initdb -D ./data --wal-segsize=1024`
- Synchronous Commit
  - What components are you testing?
- MySQL
  - `innodb_flush_log_at_trx_commit=0/1`
- PostgreSQL
  - `wal_level = minimal/replica`
  - `synchronous_commit = off/on`
Loading Database Client Libraries

- HammerDB needs access to client libraries to load interface
- CLI librarycheck command

  hammerdb> librarycheck
  Checking database library for MySQL
  Success ... loaded library mysqltcl for MySQL
  Checking database library for PostgreSQL
  Success ... loaded library Pgtcl for PostgreSQL

- Export LIBRARY_PATH

  export LD_LIBRARY_PATH=/opt/postgresql-13.2/pgsql/lib/:$LD_LIBRARY_PATH

- Use ldd on HammerDB interfaces to verify library used
  - Up to v4.1 for MariaDB you need the MySQL client, from v4.2 MariaDB clients used

  ldd libmysqltcl3.052.so
  libmysqlclient.so.21 => /usr/lib/x86_64-linux-gnu/libmysqlclient.so.21
  libpgtcl2.1.1.so
  libpq.so.5 => /opt/postgresql-13.2/pgsql/lib/libpq.so.5 (0x00007f0e20ce5000)
Schema Build
Schema Build

- **Schema Build Options**
  - Select options from menu
  - Configure with CLI commands
  - Same schema is built

- **Options vary per database**
  - MySQL Storage Engines
  - Partitioning
  - PostgreSQL stored procedures or functions

- **Key Factors in Build Performance**
  - CPU cores in client
  - I/O throughput on database
Schema Build Choices

- **Schema Build**
  - Creates tables
  - Creates and loads data
  - Creates Indexes
  - Creates functions/stored procedures
  - Gathers statistics

- **Number of Warehouses**
  - Define according to system scale
  - Entire schema scaled based on warehouse count

- **Stored Procedures**
  - New Order
  - Payment
  - Delivery
  - Stock Level
  - Order Status

- **Virtual Users to Build Schema**
  - Schema creates and loads data in parallel
  - Use number of CPU cores/threads on HammerDB client
How many warehouses?

- Default Configuration
  - Virtual Users chooses a home warehouse at random
  - 90% of the workload satisfied from the home warehouse
    - Regardless of the number configured
    - Hot and cold data
- Configure enough warehouses to ensure an even spread of Virtual Users (e.g., 4X expected VU count)
- Overprovisioning warehouses will not increase performance or scalability
- Example on 2 socket 1000 warehouses
  - Takes 8-9 minutes to load
  - Depends on CPU and Disk and Virtual Users to build schema
- Warehouse Count Limits
  - 5000 warehouses in GUI
  - 30,000 in Datagen
  - No actual limit for advanced users, only interface limits
Running the Build

- Build Schema Command
  - GUI Build Option
  - CLI buildschema command

CLI Script

```
    dbset db pg
    dbset bm TPC-C
    diset tpcc pg_count were 20
    diset tpcc pg_num_vu 4
    diset tpcc pg_superuser steve
    diset tpcc pg_superuserpass postgres
    buildschema
```
Datagen

- GUI or CLI
  - Bulk Loads to bypass database logging and network overhead

Schema Data in text files
Running the Test
Running the Test

- Driver Script Options
  - Test Loads a driver script
  - Options modify script loaded

- Test Script
  - Simple run
  - Small number of Virtual Users
  - Verify Schema Build

- Timed Script
  - Measured Test
  - Small to larger number of Virtual Users
  - Suppressed Output
Driver Script Options

- **Connection Parameters**
- **Driver Script**
- **Total Transactions**
  - Sets an upper limit for number of transactions for each Virtual User to run
- **Rampup Time**
  - Time for data to load into cache
- **Test Duration**
  - Timed period of test
- **Advanced Options**
Virtual Users

- Configure and Create Virtual Users
  - Virtual Users run in parallel
  - Each Virtual User is OS thread
  - Runs independently
Running the Test

- Click Run to start
- Transaction Mix
  - New Order 45%
  - Payment 43%
  - Delivery 4%
  - Stock Level 4%
  - Order Status 4%
- Status shown of Virtual Users
  - Running
  - Complete
  - Error Status
- Press Stop to terminate Virtual Users
Running a Test with the CLI

- Choose Options
- Load Script
- Workload is identical to that run by GUI (driver script is the same)

dbset db mysql
diset connection mysql_host localhost
diset connection mysql_port 3306
diset connection mysql_socket /tmp/mysql.sock
diset tpcc mysql_user root
diset tpcc mysql_pass mysql
diset tpcc mysql_driver timed
diset tpcc mysql_rampup 2
diset tpcc mysql_duration 5
loadscript
vuset vu 4
vucreate
vurun
runtimer 500
vudeestroy
Review Results

- Test Result Printed when compete
  - GUI
  - CLI
- Review Engine Throughput with Transaction Counter

13680 tpm
Understanding Results: NOPM vs TPM

- NOPM
  - How fast you are going
  - Close relation to official tpmC
- TPM
  - How hard your engine is working
- Comparing performance
  - NOPM can be compared between engines
  - TPM can only be compared across the same engine
  - TPM useful engineering metric to compare statistics
GUI Automation: Autopilot

- GUI Automation
- Run Unattended Test Sequence
- Define sequence of tests
  - Increased Virtual User Count
- Log Output
CLI Automation: Scripting

- CLI supports full TCL syntax
- Simple foreach loop for test sequence
- Can modify any parameters desired
- Log output

```cli
set db mysql
set connection mysql_host localhost
set connection mysql_port 3306
set connection mysql_socket /tmp/mysql.sock
set tpcc mysql_user root
set tpcc mysql_pass mysql
set tpcc mysql_driver timed
set tpcc mysql_rampup 2
set tpcc mysql_duration 5
loadscript
foreach z {1 2 4 8 12 16 20 24} {
    puts "$z VU TEST"
    vuset vu $z
    vucreate
    vurun
    runtime 500
    vudeestroy
}
Performance Profiles

- **Run Multiple Tests**
  - Increasing Virtual User Load

- **Example 56 (2 x 28) cores**

- **(Near) Linear Scale**
  - Up to CPU cores/threads
  - Dependence on Database software

- **Performance Plateau**
  - Capture Peak Performance
  - Highest CPU Utilisation

- **Contention**
  - Increasing response times
  - Flat to lower performance
Comparing Performance

- Different Systems have different profiles
  - Not predictable on CPU Count
  - Database engines differ

- MySQL example
  - Linear scale is the same
  - Sys 1 + 3 extended performance plateau
  - Sys 2 + 4 show contention earlier

- Plan for differing levels of capacity
Advanced Testing Features
Advanced Testing Features

- **Use All Warehouses**
  - Increase physical I/O to the data area

- **Connect Pooling**
  - Direct parts of the workload to different nodes in the same cluster
  - For example read/write and read-only nodes

- **Event Driven Scaling**
  - Co-routine based
  - Implements keying and thinking time
  - Scales to thousands of sessions

- **Time Profiling**
  - Capture Virtual User response times

- **Step Workloads**
  - Variable throughput by adding and removing Virtual Users

- **Advanced Features not mutually exclusive**
  - Can use some or all of the advanced features at the same time
Use All Warehouses

- All Warehouses divided between Virtual Users
  - New warehouse selected per transaction
  - More physical I/O
Connect Pooling for Clusters

- **Define in XML Configuration**
  - Multiple connections instances in cluster
  - Which transactions are directed to which nodes
  - Policy on how to allocate transactions across pool of connections e.g. round robin

- **Example RW/RO nodes**
  - Define RW transactions to primary and RO to standby

- **Reports NOPM and TPM from Primary**
  - Also reports client side TPM
  - Detailed view of transactions processed per node
Event Driven Scaling

- Default Workload is Cached
- Scaled Workloads
  - Large Session Counts
  - Fixed Throughput
  - Keying and Thinking Time delays
- Requires larger storage and networking
  - Requires middleware
  - HammerDB connects to middleware
  - Middleware connects to database
- Multiple Sessions per Virtual User
  - Uses co-routines to make key and think asynchronous
  - Appx 1 NOPM per session
- Example 1000 warehouses
  - 10,000 Sessions
  - 10,000 NOPM
Time Profiling for Response Times

- 2 Time Profiling Packages
  - Xtprof – all virtual users
  - Etprof – first active virtual user

- Xtprof
  - Profile of all virtual user response times
  - Summary of all virtual users

---

<table>
<thead>
<tr>
<th>Virtual User</th>
<th>Iterations</th>
<th>Complete</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Database</th>
<th>CALLS</th>
<th>MIN</th>
<th>AVG</th>
<th>MAX</th>
<th>TOTAL</th>
<th>RATIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle</td>
<td>253013</td>
<td>0.414ms</td>
<td>1.375ms</td>
<td>121.556ms</td>
<td>347988.879ms</td>
<td>48.514%</td>
</tr>
<tr>
<td>SQL Server</td>
<td>4.628ms</td>
<td>5.932ms</td>
<td>10.929ms</td>
<td>1951.917</td>
<td>26340.44</td>
<td>70.185%</td>
</tr>
<tr>
<td>MySQL</td>
<td>252926</td>
<td>0.183ms</td>
<td>0.822ms</td>
<td>123.252ms</td>
<td>208142.667ms</td>
<td>690.185%</td>
</tr>
<tr>
<td>POSTGRES</td>
<td>9.319ms</td>
<td>7.967ms</td>
<td>4.789ms</td>
<td>2352.499</td>
<td>3449.199</td>
<td>9.319%</td>
</tr>
<tr>
<td>POSTGRES</td>
<td>25061</td>
<td>0.589ms</td>
<td>2.663ms</td>
<td>34.680ms</td>
<td>66813.642ms</td>
<td>9.319%</td>
</tr>
<tr>
<td>POSTGRES</td>
<td>25412</td>
<td>0.699ms</td>
<td>1.782ms</td>
<td>41.734ms</td>
<td>45289.974ms</td>
<td>9.319%</td>
</tr>
<tr>
<td>POSTGRES</td>
<td>4.685ms</td>
<td>5.932ms</td>
<td>10.929ms</td>
<td>1951.917</td>
<td>26340.44</td>
<td>70.185%</td>
</tr>
<tr>
<td>POSTGRES</td>
<td>25286</td>
<td>0.149ms</td>
<td>0.696ms</td>
<td>123.539ms</td>
<td>17618.867ms</td>
<td>6.314%</td>
</tr>
</tbody>
</table>

---

![Diagram](image.png)

- PostgreSQL TPROC-C Timed Mode: Local Row: 0.0
Variable Step Workloads

- Creates Variable Load
  - Define in XML
  - Pyramid of HammerDB Instances
- Runs in CLI only
  - Primary Instance of HammerDB
  - Replica instances created automatically and connect to primary
  - Timed delay of replica starts
- Start with “steprun” command
- Measure response times
  - Variations in performance
Performance Monitoring
CPU & Database Metrics

- HammerDB Graphical CPU Monitor
- Visualization of CPU load
- eg 50% CPU Average
  - Could be 50% of all cores at 50%
  - Could be 50% of cores at 100% and 50% at 0
- Detect CPU imbalance
- System & User CPU Utilization
- Identify Interrupt bottlenecks on individual cores
- GUI Database Metrics for PostgreSQL in progress
MySQL

- MySQL 8.0.20+ recommended
  - Improved Lock Scheduling
- Monitor InnoDB storage engine
- Innotop
PostgreSQL

- PostgreSQL 13+ recommended
  - Improved throughput

- `pg_stat_statements` / `pg_sentinel`

```
% | AAS  | backend_type     | wait_event_type | wait_event
---+-------+-----------------+-----------------+------------------
48 | 28.00 | client backend  | CPU             | CPU
12 | 6.82  | client backend  | LWLock          | XactSLRU
11 | 6.18  | client backend  | LWLock          | WALInsert
9  | 5.41  | client backend  | IPC             | ProcArrayGroupUpdate
6  | 3.71  | client backend  | Client          | ClientRead
6  | 3.65  | client backend  | IPC             | XactGroupUpdate
5  | 2.82  | client backend  | Lock            | extend
2  | 0.94  | client backend  | LWLock          | ProcArray
1  | 0.35  | client backend  | IPC             | CheckpointDone
```
HPE LinuxKI

- **LinuxKI Toolset**
  (Trace-based performance analysis tool)

- **System level analysis**
  - Beyond database only statistics

MySQL - HammerDB

### 1.3.3 Trace Events of Top 10 Processes

<table>
<thead>
<tr>
<th>Freq</th>
<th>Percent</th>
<th>Trace_type</th>
<th>ElapsedT</th>
<th>Max</th>
<th>Ave</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1898956</td>
<td>43.88%</td>
<td>sys_enter</td>
<td>0.713</td>
<td>0.0044</td>
<td>0.000001</td>
<td>0</td>
</tr>
<tr>
<td>1307788</td>
<td>30.22%</td>
<td>sched_yield</td>
<td>3.808</td>
<td>0.0037</td>
<td>0.000008</td>
<td>1417</td>
</tr>
<tr>
<td>469256</td>
<td>10.84%</td>
<td>futex</td>
<td>9.256</td>
<td>0.0067</td>
<td>0.000076</td>
<td>0</td>
</tr>
</tbody>
</table>

MySQL - Sysbench

### 1.3.3 Trace Events of Top 10 Processes

<table>
<thead>
<tr>
<th>Freq</th>
<th>Percent</th>
<th>Trace_type</th>
<th>ElapsedT</th>
<th>Max</th>
<th>Ave</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>742430</td>
<td>41.55%</td>
<td>sys_enter</td>
<td>0.578</td>
<td>0.0008</td>
<td>0.000001</td>
<td>148226</td>
</tr>
<tr>
<td>444924</td>
<td>24.90%</td>
<td>recvfrom</td>
<td>0.365</td>
<td>0.0005</td>
<td>0.000002</td>
<td>0</td>
</tr>
<tr>
<td>149126</td>
<td>8.35%</td>
<td>sched_yield</td>
<td>5.559</td>
<td>0.0055</td>
<td>0.000038</td>
<td>0</td>
</tr>
</tbody>
</table>
Next Steps
Analytic Testing

- TPROC-H for Analytics
- Cloud Queries
- Stream of 22 Complex Queries
- PostreSQL Parallel Query
- Columnstores
- More complex skillset required
Published Benchmarks

- Has someone already done a performance study you can use?
  - [https://www.hammerdb.com/benchmarks.html](https://www.hammerdb.com/benchmarks.html)
- Have you published your findings for other people to use?
- Making database performance data open source benefits all
MariaDB as separate database

- TPROC-C and TPROC-H
- Current support requires MySQL client library
- Future support MariaDB client
- Opportunity to diversify workload
Contribute to HammerDB on GitHub

- Contribute to HammerDB
- All source code open source
- Documentation open source
  - Docbook format
  - Edit with any XML editor
- Issues
- Discussions
- Binary releases
Thanks!

Any questions?
THANK YOU!