Accelerating MySQL with JIT Compilers

David Yeager
david@dynimize.com

 Dynimize

Percona Live Santa Clara
April 2018
What is a Just-In-Time Compiler?

Java source code → Java Compiler → Bytecode → Java JIT Compiler → Machine code

C/C++ source code → C Compiler → Profiling Information → Dynimizer JIT Compiler → Machine code
How MySQL benefits from JITs

MariaDB 10.2 + Sysbench

OVH public cloud, 2 vCores x 2.3 Ghz (Broadwell Xeon) template B2-7, BHS1 datacenter
How MySQL benefits from JITs

MySQL 5.7
tpcc-mysql / Wordpress

*OVH public cloud, 2 vCores x 2.3 Ghz (Broadwell Xeon) template EG-7-SSD, BHS1 datacenter
*tpcc-mysql is not validated or certified by the TPC corporation and so this is not an official TPC-C result
Dynimizer Usage In a Nutshell

Installation

$ sudo bash -c 'bash <(wget -O - https://dynimize.com/install) -default'

Usage

$ sudo dyni -start
Dynimizer started

$ sudo dyni -status
Dynimizer is running
mysqld, pid: 20722, dynimized

$ sudo dyni -status
Dynimizer is running
mysqld, pid: 20722, dynimized
Dynimizer Usage

1. Start
   
   `$ sudo dyni -start`
   
   Dynimizer started

2. Monitoring
   
   `$ sudo dyni -status`
   
   Dynimizer is running

3. Profiling
   
   `$ sudo dyni -status`
   
   Dynimizer is running
   mysqlld, pid: 20722, profiling

   Reoptimize (can be disabled)

4. Dynimizing
   
   `$ sudo dyni -status`
   
   Dynimizer is running
   mysqlld, pid: 20722, dynimizing

5. Dynimized
   
   `$ sudo dyni -status`
   
   Dynimizer is running
   mysqlld, pid: 20722, dynimized

   pid 20722
   drastically change phase?

   N
   Y

   Reoptimize (can be disabled)
Hardening Dynimizer For Production

$ dyni -optimizeOnce:y

Default is to reoptimize after large changes in workload

- This setting disables it
- Prevents temporary performance overhead if had to re-optimize in middle of a workload
- No changes to machine code == more stable
- More conservative
- If workload changes drastically, Dynimizer improvement will be reduced
Hardening Dynimizer For Production

$ dyni -secureCodeCache:y

Default code cache is executable, readable and writable at the same time

- This setting makes code cache executable and read-only
- Enable automatically on SELinux for extra security
- You may want this enabled regardless

$ dyni -pid <number>

You may want to limit Dynimizer to a specific mysqld process
Configuring with /etc/dyni.conf

This is dyni.conf after default installation

Overridden by command-line options
- For example:
  
  ```
  $ dyni -optimizeOnce:y
  ```
  
  will override dyni.conf

Can target other programs by adding exe names under [exeList]
- Non-mysql targets not supported yet so test thoroughly!
Sources of performance gain

OLTP workloads are mostly front-end CPU stalls

- Instruction cache misses, branch mispredictions, ITLB misses
- Use profiling information to better layout the machine code, reduce branching

Other profile guided optimizations

- Hot call-site inlining, sparse conditional constant propagation
- Dead code elimination, copy propagation
- Loop unrolling, branch target alignment
- Other optimizations
When can Dynimizer help?

Most Beneficial

- High CPU usage
- Long running workloads
- Well indexed queries
- Have fully optimized MySQL, want even more performance
- Read heavy workload
- SELECT: lots of front-end CPU stalls
- Working set fits into the buffer pool

Least Beneficial

- Low CPU usage scenarios
- Lots of writes to slow disks
  - IO bottleneck
- Working set doesn't fit in buffer pool
- Full table scans
- Short mysqld process lifetime
- > 5 k threads
  - Current ptrace scales poorly

Most Beneficial

Least Beneficial
When can Dynimizer help?

```bash
$ perf stat -e r0280:u,r0380 -p 30041 sleep 30
Performance counter stats for process id '30041':

3,224,918,396 r0280:u [100.00%]
39,530,772,359 r0380
```

- I-cache misses a good indicator
- r0280 means I-cache misses for last several generations of Intel CPUs
- u: is user mode, r0380 is instruction fetches
- \(\frac{3,224,918,396}{39,530,772,359} = 8\%\)
- > 5% indicates instruction bandwidth is a serious bottleneck
Dynimizer is the Everyman's PGO

Profile Guided Optimization

- Available in GCC
  - Compile with instrumentation
  - Training run with profiling
  - Recompile
- Difficult to find a representative workload that will stand up over time
- Labour intensive
- For large scale MySQL deployments that can amortize the labour

Dynimizer JIT

- Orders of magnitude easier
  - Trivial usage: $ dyni -start
  - Not required to build from source
  - 1-5 minutes to optimize
- Zero downtime
- Includes shared libraries
- Way more flexible
  - Can optimize code for each run
## Supported Targets

<table>
<thead>
<tr>
<th>Optimization Target</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL Server</td>
<td>5.5 – 5.7</td>
</tr>
<tr>
<td>MariaDB Server</td>
<td>5.5 – 10.2</td>
</tr>
<tr>
<td>Percona Server</td>
<td>5.5 – 5.7</td>
</tr>
</tbody>
</table>

- Linux x86-64
- That means **mysqld**
Sysbench: MySQL 5.7 OLTP-RO

CPU: Intel(R) Xeon(R) CPU E3-1270 v6 @ 3.80GHz, 4 cores, 8 Threads (Kaby Lake)
RAM: 32 GB of 2400 MHz DDR4

*This is a dedicated server rented from OVH, model: SP-32 Server, data center BHS 5
*Relative speedups the similar across various table size or number of tables, so long fits into memory
Sysbench: MySQL 5.7 OLTP Simple

![Graph showing Transactions/Second vs Threads with and without Dynimizer]

- WITH Dynimizer
- WITHOUT Dynimizer

Dynimize
Sysbench: TPS Increase

![Graph showing TPS increase across different threads and workloads.]

- oltp read-only
- oltp-simple
- select
- select-random-ranges
Reduction in Branch Mispredictions
Reduction in ITLB Misses

![Graph showing reduction in ITLB misses with different thread counts and workloads. The x-axis represents the number of threads ranging from 1 to 128, and the y-axis represents the percentage reduction in ITLB misses. The graph includes lines for 'oltp read-only', 'oltp-simple', 'select', and 'select-random-ranges', each indicating a different workload scenario. The graphs show varying degrees of reduction depending on the workload and the number of threads.]

Dynimize
Reduction in I-Cache Misses

![Graph showing reduction in I-Cache Misses across different thread counts for various database workloads: oltp read-only, oltp-simple, select, select-random-ranges. The graph demonstrates varying degrees of reduction in cache misses from 1% to 60% as the number of threads increases from 1 to 128.](image-url)
Increase in Instructions Per Cycle

Threads
oltp read-only
oltp-simple
select
select-random-ranges

Dynimize
Caveats: Steep warmup curve

Will be reduced in next major release
Caveats: Memory Usage

- 4 GB per process during the **dynimizing** phase only
  - Freed once optimized
  - Extra RAM not necessary. Just increase swap by 4 GB
  - May not be appropriate for some micro cloud instances

- Will be reduced in next major release.
Noteworthy attributes

- Exploiting Run-Time Information
- Zero downtime
- Optimize in minutes
- Target app source code not required
- Optimize across shared libraries
- Simple usage
- Little to no configuration necessary
Coming soon...

- Cache compilation for instant optimized restart of target processes (mysqld)
- Lower profiling and memory overheads
- Improved phase change detection
- More optimizations
- Toggle between code cache versions depending on program phase
- Many more target programs to optimize.
  - Have observed similar improvements with MongoDB
- Many new optimizations and speedups along the way
Questions?

To learn more visit dynimize.com
Rate My Session

Introducing gh-ost: triggerless, painless, trusted online schema migrations

11:20 - 12:10, Matterhorn 2

Tap a star to rate

Feedback (optional)

Anonymous

Submit