

libdrizzle

Eric Day

eday@oddments.org
<http://www.oddments.org/>

Overview

- What is libdrizzle?
- Client example
- Result buffering
- Introduction to non-blocking I/O
- How libdrizzle uses non-blocking I/O
- Revisit client example
- Project status
- New protocol
- Feedback!

Wait, what is Drizzle?

- Time to crawl out from under the rock...
- Derived from MySQL source code
- Reworking of many (all?) parts
- Focus on micro-kernel design (plugins!), multi-core hardware (think 64+), and “The Cloud”
- <http://drizzle.org/>
- <https://launchpad.net/drizzle>
- <http://planetdrizzle.org/>

libdrizzle

- The derived MySQL client library needed an overhaul, easier to start from scratch
- Written in straight C, but other language interfaces are being built
- Rethink I/O and result buffering
- Concurrency
- Improve the developer experience through an intuitive interface

libdrizzle

- The new libdrizzle was born
- Complete client and low level protocol library
- C - <https://launchpad.net/libdrizzle>
- PHP- <https://launchpad.net/drizzle-php-ext>
- Monty Taylor - SWIG (Perl, python, Ruby, ...)
<https://launchpad.net/drizzle-interface>
- Class interfaces, DBD, PDO, ... coming soon
- Supports MySQL protocol
- Enough talk, code!

```
# Create library handle.
$drizzle= drizzle_create();

# Run 1st query.
$con= drizzle_con_add_tcp($drizzle, "server1", 3306,
                           "user", "pass", "db", 0);
$result= drizzle_query($con, "SELECT ...");
drizzle_result_cache($result);

# Use result.

# Run 2nd query.
$result= drizzle_query($con, "SELECT ...");
drizzle_result_cache($result);

# Use result.

# Run 3rd query on a different server.
$con= drizzle_con_add_tcp($drizzle, "server2", 3306,
                           "user", "pass", "db", 0);
$result= drizzle_query($con, "SELECT ...");
drizzle_result_cache($result);

# Use result.
```

Query Result Buffering Options

- Don't buffer anything, use data directly from buffer used for read()
- Buffer at the field data level
- Buffer at the row level
- Buffer the entire result
- Or mix and match, can switch between modes by using another function (great for BLOBs)

```
lap> valgrind ./client -d "sakila" "SELECT name FROM category" | wc -l
...
==9361== malloc/free: in use at exit: 0 bytes in 0 blocks.
==9361== malloc/free: 4 allocs, 4 frees, 37,340 bytes allocated.
...
64
```

```
lap> valgrind ./client -d "sakila" "SELECT * FROM film JOIN actor" | wc -l
...
==9367== malloc/free: in use at exit: 0 bytes in 0 blocks.
==9367== malloc/free: 4 allocs, 4 frees, 37,340 bytes allocated.
...
3800192
```

Non-blocking I/O

- Normally, `read()` and `write()` will block until done
- Enable non-blocking with `fcntl()` `O_NONBLOCK`
- Now `read()` and `write()` do whatever they can and return `EAGAIN` when it would block
- Block for I/O on multiple file descriptors using `poll()`, `select()`, ...
- Partial reads and writes possible
- You need to keep more state

Non-blocking I/O in libdrizzle

- All I/O is actually non-blocking
- Uses poll(), will optionally use libevent soon
- Flag in library handle lets user control if the interface is blocking or not
- Allows for concurrent connections
- ... and queries!

```
# Create library handle.
$drizzle= drizzle_create();

# Run 1st query.
$con= drizzle_con_add_tcp($drizzle, "server1", 3306,
                           "user", "pass", "db", 0);
$result= drizzle_query($con, "SELECT ...");
drizzle_result_cache($result);

# Use result.

# Run 2nd query.
$result= drizzle_query($con, "SELECT ...");
drizzle_result_cache($result);

# Use result.

# Run 3rd query on a different server.
$con= drizzle_con_add_tcp($drizzle, "server2", 3306,
                           "user", "pass", "db", 0);
$result= drizzle_query($con, "SELECT ...");
drizzle_result_cache($result);

# Use result.
```

```
# Construct queries.
$query[]= "SELECT ...";
$query[]= "SELECT ...";
$query[]= "SELECT ...";

# Create library handle.
$drizzle= drizzle_create();

# Create connections.
$con[]= drizzle_con_add_tcp($drizzle, "server1", 3306,
                            "user", "pass", "db", 0);

$con[]= $con[0];
$con[]= drizzle_con_add_tcp($drizzle, "server2", 3306,
                            "user", "pass", "db", 0);

# Run queries.
for ($x= 0; $x < 3; $x++)
{
    $result[$x]= drizzle_query($con[$x], $query[$x]);
    drizzle_result_cache($result[$x]);
}

# Use results.
```

```
# Construct queries.
$query[]= "SELECT ...";
$query[]= "SELECT ...";
$query[]= "SELECT ...";

# Create library handle.
$drizzle= drizzle_create();

# Create connections.
$con[]= drizzle_con_add_tcp($drizzle, "server1", 3306,
                             "user", "pass", "db", 0);
$con[]= drizzle_con_clone($drizzle, $con[0]);
$con[]= drizzle_con_add_tcp($drizzle, "server2", 3306,
                             "user", "pass", "db", 0);

# Build query list and run.
for ($x= 0; $x < 3; $x++)
    $query_list[]= array($con[$x], $query[$x]);

$result= drizzle_query_list($drizzle, $query_list);

# Use results.
```

Project Status

- Still under heavy development
- Interfaces could still change
- Higher level interfaces still being developed
- PHP Extension - Not all functions are done yet
- Client side utilities being converted
- Server protocol interface will also be replaced

New Drizzle Protocol

- Two layers of packets
- Sharding key in lower layer packet
- Read-only hints for sharding
- UDP support for some queries
- Packets broken down into chunks
- Authentication optional
- Checksums
- Concurrent queries on a single connection

Feedback!

What would you like to see?