



Handlersocket

Ryan Lowe
Percona Live London 2011

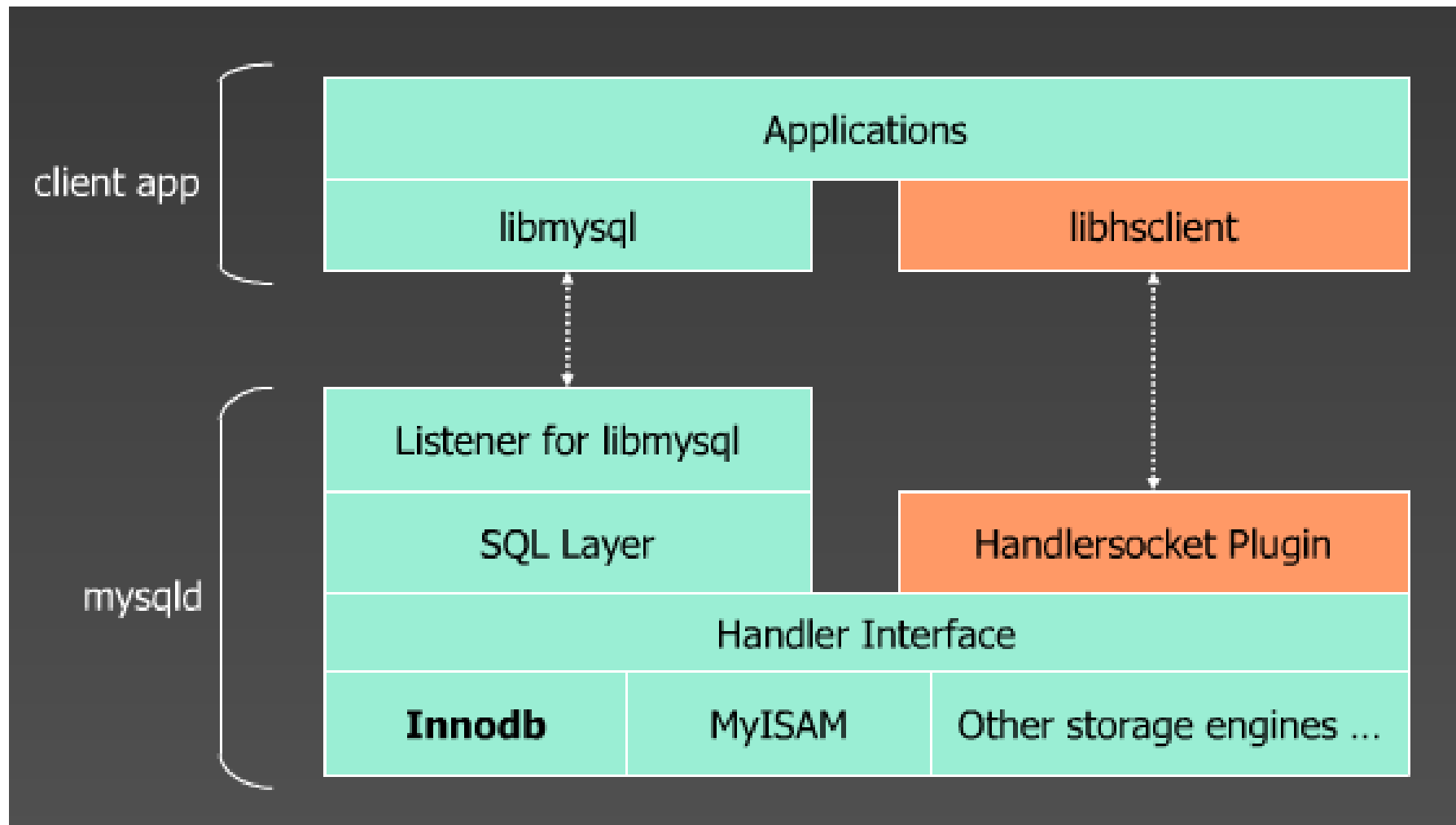
Agenda

- What is Handlsocket
- mysqlds vs libhsclient
- How much does it save?
- How is it used?
- When can it be used?
- Other nosql alternatives for MySQL

What is Handlsocket

- Daemon plugin for MySQL providing direct (nosql) access to Innodb tables

What is Handlsocket



Why use Handlsocket

- Simple CRUD commands
- Bypass SQL layer of MySQL
- Use the right tool for the right job
- Keep your application stack simpler
- Performance

Architecture

LAMP → GLANMRMD[P|P|P|R|J]

mysql vs libhsclient

- **mysqld:**
 - SQL Queries, multiple storage engines. Usual overhead from parsing, priv check, global mutexes, etc.
- **libhsclient**
 - Get/Set/Find queries. Innodb only.

Savings?

- SQL Parsing
- Privilege checks
- Query cache checks
- Connection pooling/thread creation

Oprofile (1/3)

libmysql (Akira's Numbers)		
samples	%	symbol name
748022	7.7355	MYSQLParse(void*)
219702	2.2720	my_thread_fastmutex_lock
205606	2.1262	make_join_statistics(...)
198234	2.0500	btr_search_guess_on_hash
180731	1.8690	JOIN::optimize()
177120	1.8317	row_search_for_mysql
171185	1.7703	lex_one_token(void*,void*)
162683	1.6824	alloc_root
131823	1.3632	read_view_open_now
122795	1.2699	mysql_select(...)

HandlerSocket (Akira's Numbers)		
samples	%	symbol name
119684	14.7394	btr_search_guess_on_hash
58202	7.1678	row_search_for_mysql
46946	5.7815	mutex_delay
38617	4.7558	my_thread_fastmutex_lock
37707	4.6437	buf_page_get_known_nowait
36528	4.4985	rec_get_offsets_func
34625	4.2642	build_template(...)
20024	2.4660	row_sel_store_mysql_rec
19347	2.3826	btr_cur_search_to_nth_level
16701	2.0568	row_sel_convert_mysql_key_to_innobase

Oprofile (2/3)

libmysql (5.1.56)		
samples	%	symbol name
57390	2.4548	MYSQLparse(void*)
42091	1.8004	String::copy(...)
32543	1.3920	__read_nocancel
30536	1.3062	btr_search_guess_on_hash
24630	1.0535	my_wc_mb_latin1
24407	1.0440	memcpy
23911	1.0228	MYSQLlex(void*, void*)
22392	0.9578	pthread_mutex_lock
21973	0.9399	fcntl
20529	0.8781	my_utf8_uni

libmysql w/ prepared statements (5.1.56)		
samples	%	symbol name
18054	4.1415	String::copy(...)
14223	3.2627	make_join_statistics(...)
11934	2.7376	JOIN::optimize()
10140	2.3261	my_wc_mb_latin1
7152	1.6407	my_utf8_uni
7092	1.6269	Protocol::send_fields(List*, unsigned int)
6530	1.4980	JOIN::prepare(...)
6175	1.4165	Protocol::store_string_aux(...)
5748	1.3186	create_ref_for_key(...)
5325	1.2215	mysql_execute_command(THD*)

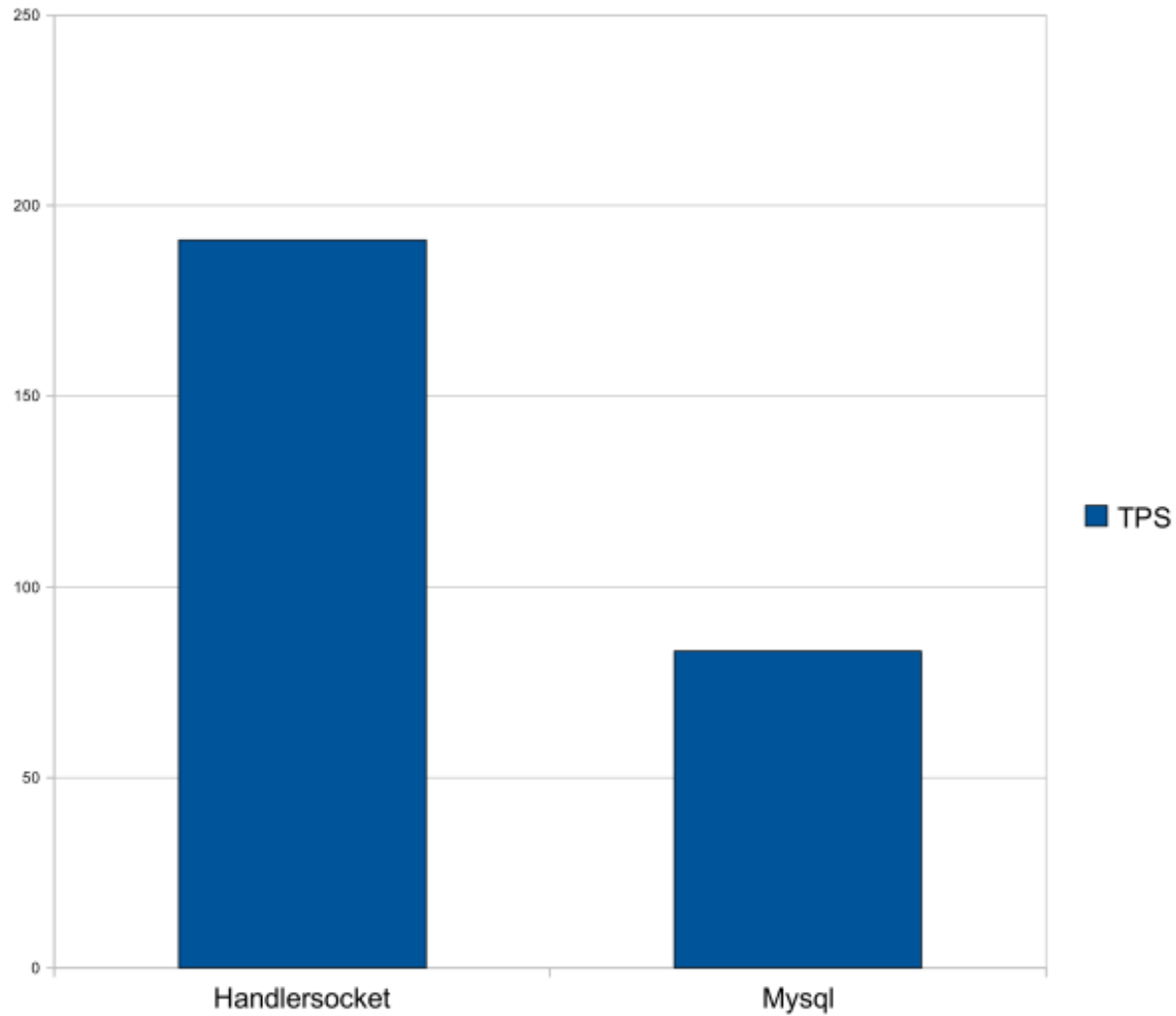
Oprofile (3/3)

HandlerSocket (5.1.56)		
samples	%	symbol name
18946	2.9918	btr_search_guess_on_hash
15853	2.5034	vfprintf
8745	1.3810	row_search_for_mysql
7021	1.1087	buf_page_get_known_nowait
5212	0.8230	__find_specmb
5116	0.8079	dena::dbcontext::cmd_find_internal(...)
5100	0.8054	build_template(...)
4866	0.7684	_IO_default_xspu
4794	0.7570	dena::hstcpsvr_worker::run_one_ep()
4538	0.7166	send

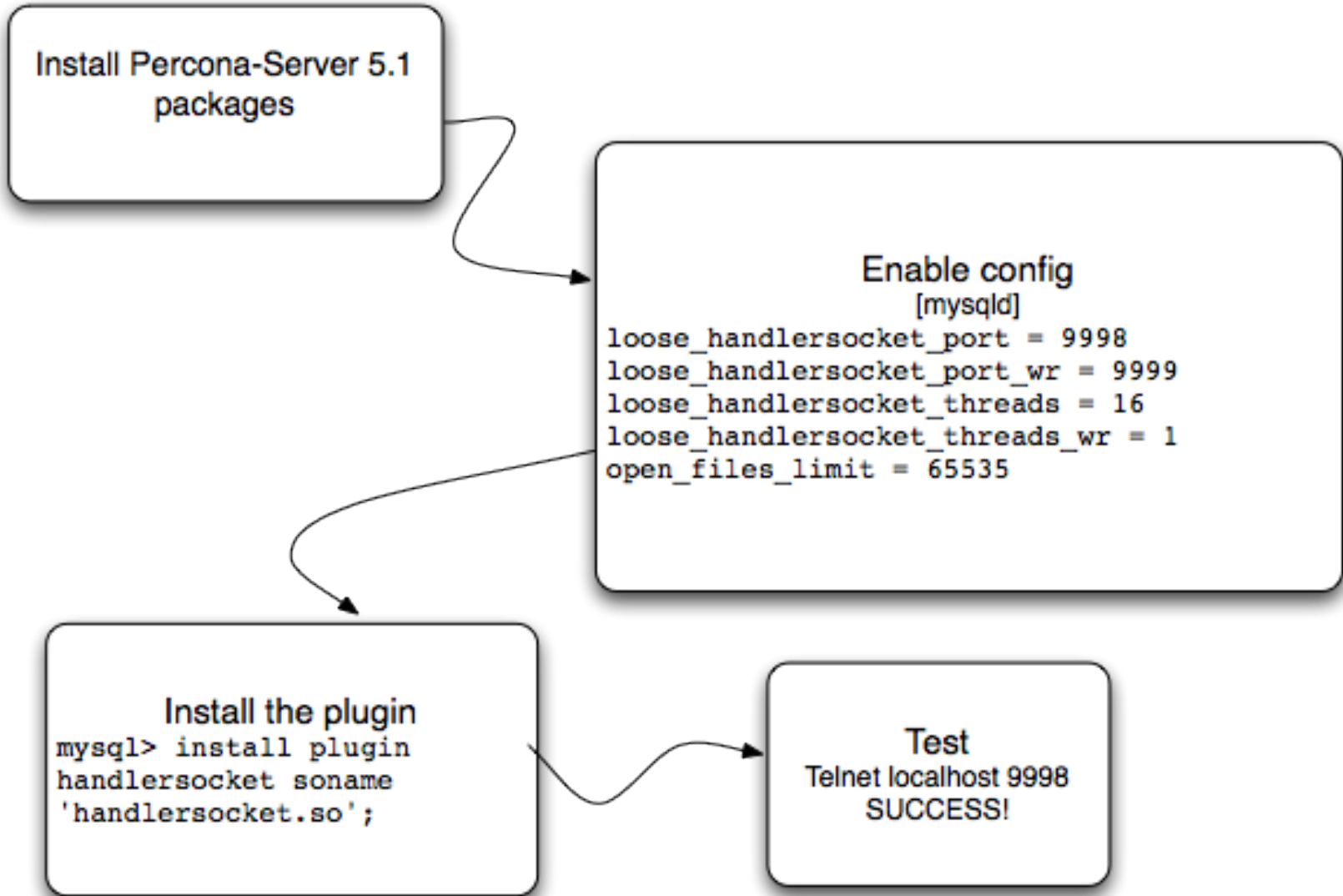
CPU Utilization

- Libmysql
 - Lots of CPU time spent in mysqld
 - Parsing SQL is slow
 - Schedule() is called frequently
- Handlersocket
 - Most CPU time is consumed in the kernel
 - Inside mysqld, Innodb consumes the most CPU
 - Schedule() is not called frequently

Encouraging example



How to install



Examples

- Simple request/response protocol
- open_index, find, find_modify, insert, auth

Examples

- All examples are in python and use the employees database (<http://bit.ly/employees-db>)

find: how to

```
from pyhs import Manager #we'll ommit from
                          #further examples
read_servers = [('inet', 'localhost', 9998)]
write_servers = [('inet', 'localhost', 9999)]
hs = Manager(read_servers, write_servers)
data = hs.find('employees', 'employees',
               ['emp_no', 'first_name'], 10001)
print dict(data)
```

find: protocol

```
%> telnet localhost 9999
```

```
Trying 127.0.0.1...
```

```
Connected to localhost.localdomain (127.0.0.1).
```

```
Escape character is '^]'.
```

```
P 0 employees employees PRIMARY emp_no,first_name
```

```
0 1
```

```
0 = 1 10001
```

```
0 2 10001 Ryan
```

```
(SELECT `emp_no`,`first_name` FROM `employees`.`employees` WHERE  
`emp_no`=10001 LIMIT 1,0;)
```

insert: how to

```
hs.insert('employees','employees',[(('emp_no','50000'),  
('first_name','Ryan'),('last_name','Lowe'))])
```

insert: protocol

0 + 3 50000 Ryan Lowe
0 1

```
INSERT INTO `employees`.`employees`  
(`emp_no`,`first_name`,`last_name`) VALUES (50000,'Ryan','Lowe');
```

secondary key find: how to

```
data = hs.find('employees','employees','=',  
['first_name','last_name'],['Ryan'],'idx_first_name')
```

secondary key find: protocol

P 0 employees employees idx_first_name first_name,last_name

0 = 1 Ryan

0 2 Ryan Lowe

```
SELECT first_name,last_name FROM `employees`.`employees` WHERE  
first_name = 'Ryan';
```

protocol summary

- find
 - Supports filtering and limit
 - Useful for paginating
- find_modify
 - 'atomic' DM statements, like increment a column.
- Insert
 - insert and find_modify statements will be batched in transactions

Configuration

- Separate read/write ports.
- Separate read/write # of threads.
- Basic network config
 - listen backlog, send and receive buffer size

Configuration

- `handlersocket_port`
- `handlersocket_port_wr`
- `handlersocket_epoll`
- `handlersocket_threads`
- `handlersocket_threads_wr`

Configuration

- handlersocket_verbose
- handlersocket_address
- handlersocket_timeout
- handlersocket_backlog
- handlersocket_sndbuf

Configuration

- `handlersocket_rcvbuf`
- `handlersocket_readsize`
- `handlersocket_accept_balance`
- `handlersocket_wrlock_timeout`

Status counters

- Hs_table_open
- Hs_table_close
- Hs_table_lock
- Hs_table_unlock

Logging

- Will not write to general- or slow- logs
- Creates RBR events in the binary log
- Will write stack traces to the error log

Limitations and gotchas

- Does not use Query Cache
- No authentication
 - Other than plain text shared secret
- No access control
- No accountability

use cases

- Single table access patterns
- Security not (that) important
- Mostly read only loads
- CPU bound workload

use cases

- High concurrency read workloads
- Inbox building
- News page
- Working set in memory read workloads
- Buffer pool warmup for non standard access patterns

alternatives

- Memcached Daemon
- NDBAPI
- HANDLER Statement
- MyCached
- PRIMEBASE Blob Streaming Engine

Percona Live London Sponsors

Platinum Sponsor



Gold Sponsor



Silver Sponsors



Percona Live London Sponsors

Exhibitor Sponsors



Friends of Percona Sponsors

Couchbase



Monty Program



Tokutek



Media Sponsors



O'REILLY®

Annual MySQL Users Conference

Presented by Percona Live

The Hyatt Regency Hotel, Santa Clara, CA

April 10th-12th, 2012

Featured Speakers

Mark Callaghan, Facebook

Jeremy Zawodny, Craigslist

Marten Mickos, Eucalyptus Systems

Sarah Novotny, Blue Gecko

Peter Zaitsev, Percona

Baron Schwartz, Percona

The Call for Papers is Now Open!

Visit www.percona.com/live/mysql-conference-2012/



ryan.a.lowe@percona.com
@Percona

We're Hiring! www.percona.com/about-us/careers/



PERCONA
LIVE

www.percona.com/live