

An Introduction to HSCALE



optivo[®]
Enterprise Email Marketing

An Introduction to HSCALE



Content

- » The problem
- » Possible ways to a solution
- » MySQL Proxy
- » HSCALE – what is it?
- » What is it NOT?
- » What's the status right now?
- » What's coming next?

The Problem



Size

- » Very HUGE tables (eem_mailingtouser currently at ~1.5 billion rows)
- » Each month ~250 million rows are added
- » Thus tables not manageable anymore – ALTER TABLE nearly impossible
- » Read a wonderful blog post by Peter Zaitsev: Small things are better
- » Need to buy bigger boxes to scale (vertically) – more disk space needed

Biggest problem: We are running out of IDs!

- » Primary key column is INT UNSIGNED to save space (especially in caches)
- » BUT can only store values up to ~4.4 billion (current max: 2.9 billion)
- » Need to change to BIGINT but ALTER TABLE is hard

The Problem - Continued



Performance

- » Performance of single MySQL server is feasible *right now*
- » But more customers with bigger datasets are added every month
- » Master/slave setup is at its limit (slave is not catching up)
- » Some customers require data to be stored separately
- » At the moment there is one big database for all

So, performance is not an issue *NOW* but *will be* in the near future!

Possible Ways to a Solution



Scale „inside“ the database (MySQL Cluster/NDB, Oracle RAC, ...)

- » Pros (in best case): No application change, „plug-in“ solution
- » Cons: There is not really a solution...
- » MySQL Cluster „is a whole different thing“
- » Oracle RAC: Expensive, application changes, needs „Oracle Guru“

Partitioning or „sharding“ your data

- » Divide the data logically
- » Split up huge tables by an attribute (i.e. mailing, mailing group)
- » Put logic into the application to choose the right „shard“
- » Pros: Fine grained data control, based on your actual needs
- » Cons:
 - Application needs to be rewritten / changed
 - Handling of „shards“ has to be implemented

Possible Ways to a Solution - Partitioning



Put partition logic inside the application's DAO layer

- » Write a `DistributedBaseDao` which selects the right connection / table
- » Extend it for all DAOs that access „sharded“ tables
- » Pros: Fine grained control, developer is aware of „sharding“ all the time
- » Cons: A lot of code, error-prone

Put partition logic outside the application into a proxy or JDBC driver

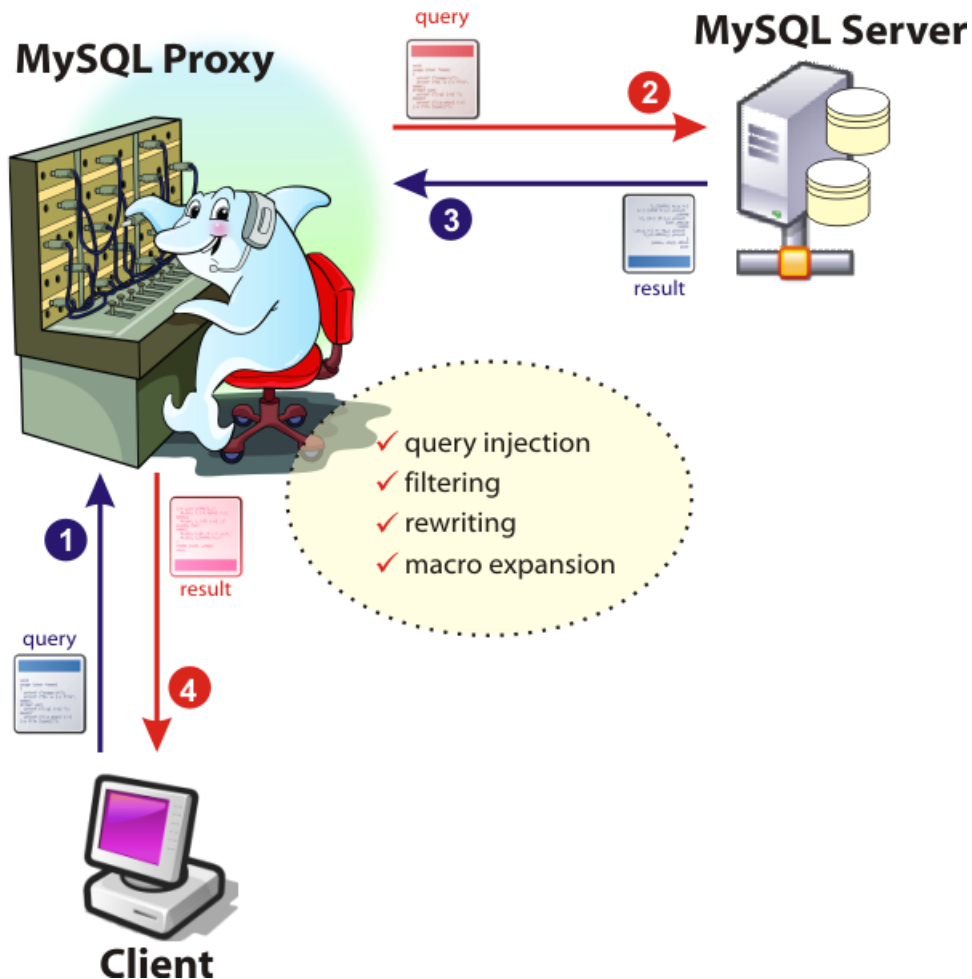
- » A proxy: Something between application and MySQL server
- » Details follow on the next pages...
- » Pros: Fewer application changes, central handling of „shards“
- » Cons:
 - Application still needs to be customized for special cases
 - Limited sub-set of SQL might be used



We (try to) go with the proxy solution!

Read a more detailed discussion of all possible solutions and their pros and cons here: [MySQL Partitioning on Application Side](#)

MySQL Proxy



Picture taken from Giuseppe Maxia's great article: [Getting Started with MySQL Proxy](#)



Some key facts

- » Developed by MySQL AB
- » Lead developer: Jan Kneschke (developer of lighttpd)
- » Current version: 0.6.1 *alpha*
- » Project home page: http://forge.mysql.com/wiki/MySQL_Proxy
- » Implements the MySQL protocol – acts like a MySQL server

Main use cases

- » Connection pooling
- » Failover
- » RW-Splitting
- » Extensible via Lua



What is Lua?

- » A light weight, high performance scripting language
- » Limited (but sufficient) feature set (procedural, modular)
- » Widely used as customization and UI language (World of Warcraft)
- » Project home page: <http://www.lua.org>

Lua within MySQL

- » Lua modules can be loaded into the proxy
- » Intercept the whole lifecycle by implementing methods:
`init(), connect_server(), read_handshake(), send_handshake(),
read_auth(), send_auth(), read_auth_result(), send_auth_result(),
read_query(), read_query_result(), send_query_result(), cleanup()`

MySQL Proxy – Lua Scripting - Example



A simple example – rewrite a query

```
function read_query(packet)
  if (string.byte(packet) == proxy.COM_QUERY) then
    local query = string.sub(packet, 2)
    print ("received " .. query)
    local replacing = false
    if string.match(string.upper(query), '^%s*SLECT') then
      query = string.gsub(query, '^%s*%w+', 'SELECT')
      replacing = true
    end
    if (replacing) then
      print("replaced with " .. query )
      proxy.queries:append(1, string.char(proxy.COM_QUERY) .. query )
      return proxy.PROXY_SEND_QUERY
    end
  end
end
```

Read an excellent introduction by Giuseppe Maxia [here](#).

HSCALE – What is it?



Put simply: A MySQL Proxy Lua module that handles partitioning

» **Transparently analyze and rewrite queries**

» **Partitioning logic moved from application to proxy module**

» **Links:**

- <http://www.hscale.org> - Project home page
- <http://aprilmayjune.org> - Personal blog with a lot of information

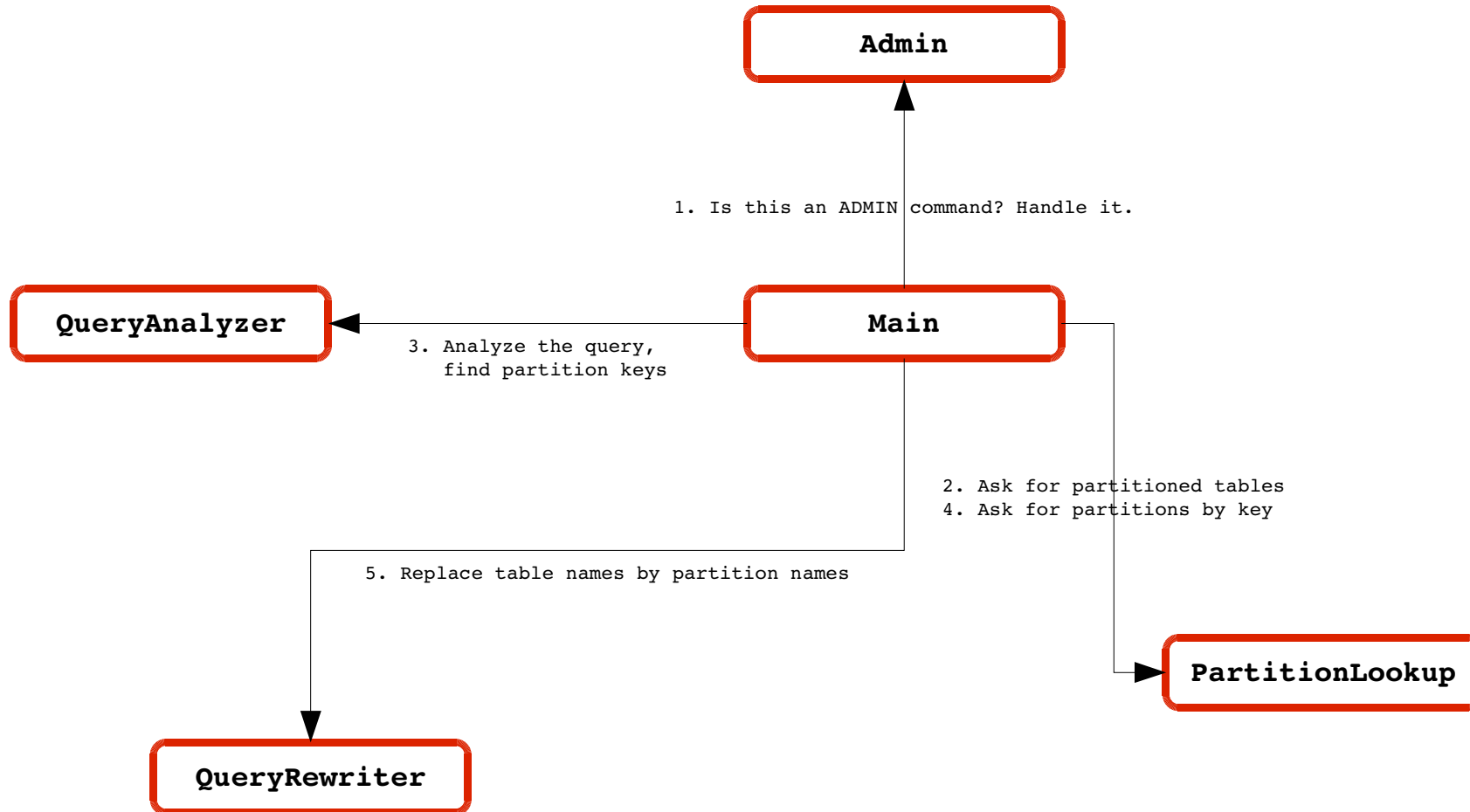
HSCALE – What is it?



Splitting up tables into partitions

- » For each table there will be multiple tables called partitions
- » Example: HSCALE is configured to split up table `eem_mailingtouser` by column `lmailing` and spread them by modulo 3
- » Instead of one `eem_mailingtouser` there are `eem_mailingtouser_0`, `eem_mailingtouser_1` and `eem_mailingtouser_2` but NO (physical) `eem_mailingtouser`
- » The application still sees the table `eem_mailingtouser` – i.e. the partitions are transparent to the application
- » `SELECT * FROM eem_mailingtouser WHERE lmailing = 1234`
will be „automagically“ rewritten to
`SELECT * FROM eem_mailingtouser_1 WHERE lmailing = 1234`

HSCALE – What is it?



HSCALE – What is it?



Handled queries

- » A lot of queries can be handled – see unit tests
- » ALTER | CREATE | DROP TABLE works

Admin commands

- » HSCALE SHOW STATUS
- » HSCALE SHOW FULL_PARTITION_SCANS

Full partition scans

- » SELECT * FROM eem_mailingtouser will send the query to all partitions (eem_mailingtouser_x) and combine the result – transparently
- » Works with LIMIT clause
- » Natural order of rows is not preserved – you should not rely on it anyway
- » Full partition scan and ORDER BY does not (and will never) work

HSCALE – What is it NOT?



HSCALE is not a „plug-in and forget“ solution

- » Not a full abstraction of a MySQL server
- » Supports only partitioning by a single column
- » HSCALE is a solution to *aid* application based partitioning
- » Thus you might still need to change aspects of your application
- » But you don't have to re-invent the wheel and implement „sharding“ logic

Current status



HSCALE 0.1 released publically

- » First version released under GPLv2 on 2008-04-10
- » Good resonance by the community

Progress on partitioning of our application

- » Partioned tables: `m2u`, `opens`, `clicks`, `responses`, `unsubscribes`
- » Integration tests (< 1,000) run against it
- » Only minor changes needed to be implemented – the idea works!
- » At a first glimpse performance seems to be ok (added 10-15% in certain test cases) – but more profiling needed

What's next?



Near future (version 1.0)

- » Main goal: Make HSCALE work for the needs of *our* application
- » More robustness: Reject/handle all queries (like `SELECT ... INTO OUTFILE` is invalid when full partition scan is used, `DESCRIBE table`)
- » Implement another partition lookup module
 - `ModulusPartitionLookup` is not feasible for real-life use cases
 - `DictionaryBasedPartitionLookup` – define which partition is where
- » With new lookup module: Implement admin commands like
 - `HSCALE CREATE PARTITION ...`
 - `HSCALE MOVE PARTITION ...`
 - `HSCALE JOIN PARTITION ...`
- » More (performance) testing!

What's next?



„Far“ future (version 2.0 and beyond)

- » Split across multiple databases / MySQL servers (XA)
 - » Allow more queries, improvements on full partition scans
 - » (Optional) Primary key constraint checking (not needed by our application)
-
- » What do you expect? ;)

Thanks!



Discussion