

Scaling Databases with DBIx::Router

Perrin Harkins
We Also Walk Dogs

What is DBIx::Router?

- Load-balancing
- Failover
- Sharding
- Transparent
 - (Mostly.)

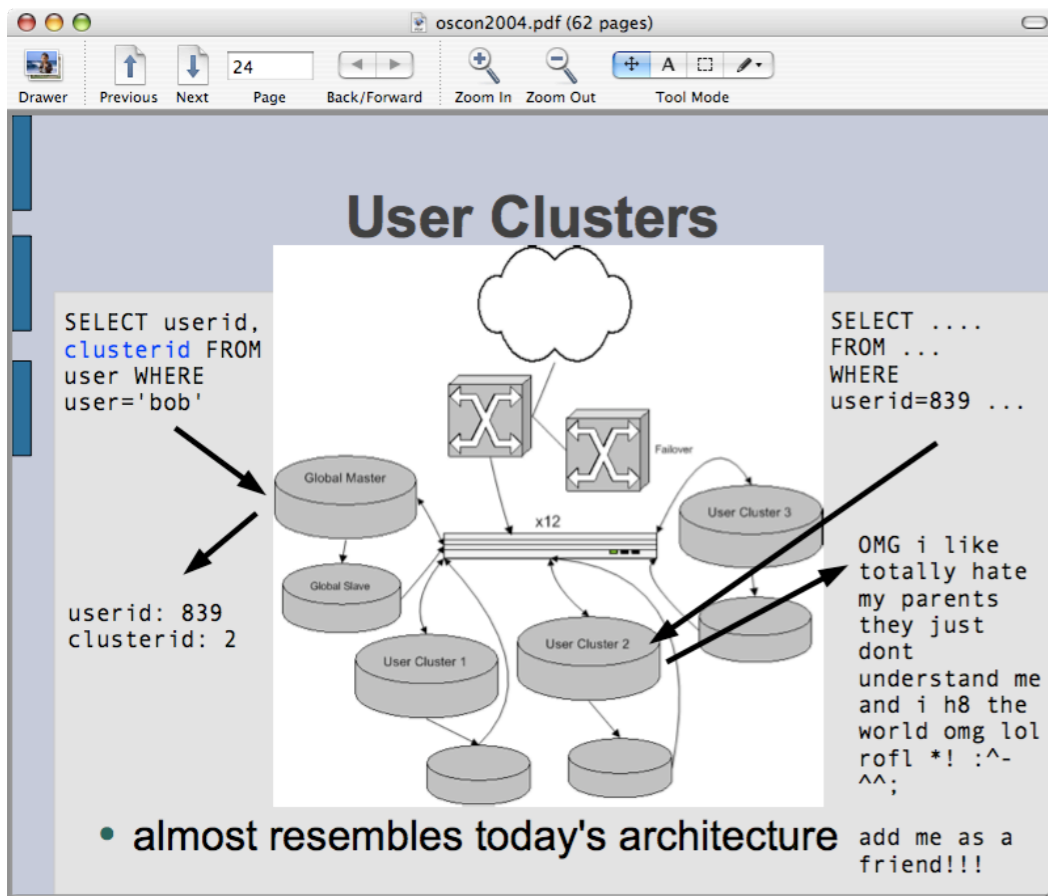
Why would you need this?

- Web and app servers are easy to scale
 - Just add another dozen boxes

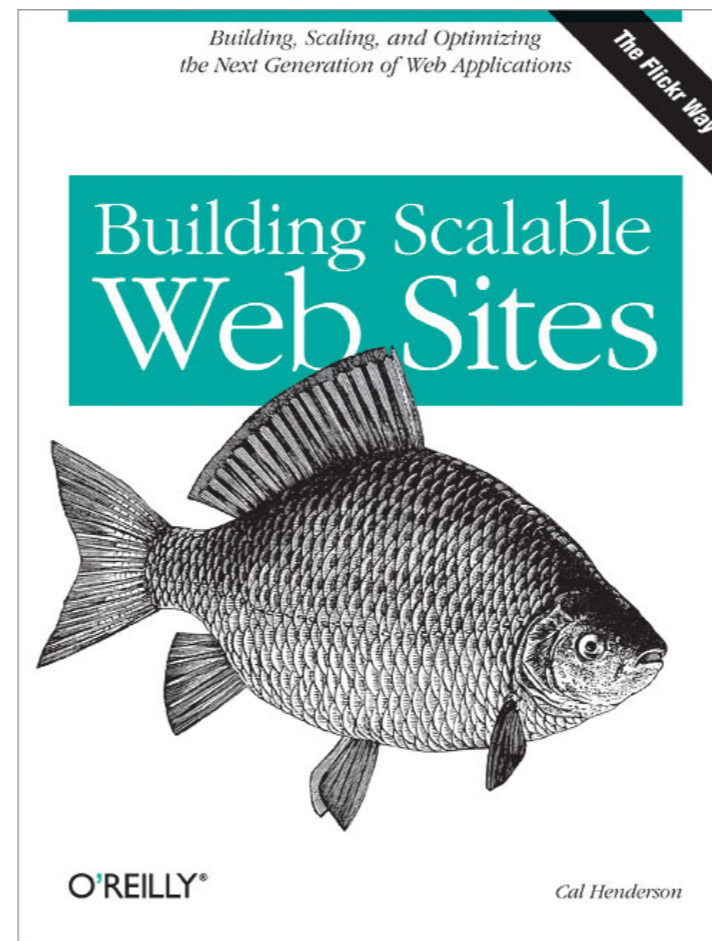
Why would you need this?

- Databases not so much
 - Big iron
 - Commercial clustering solutions
 - Human sacrifice

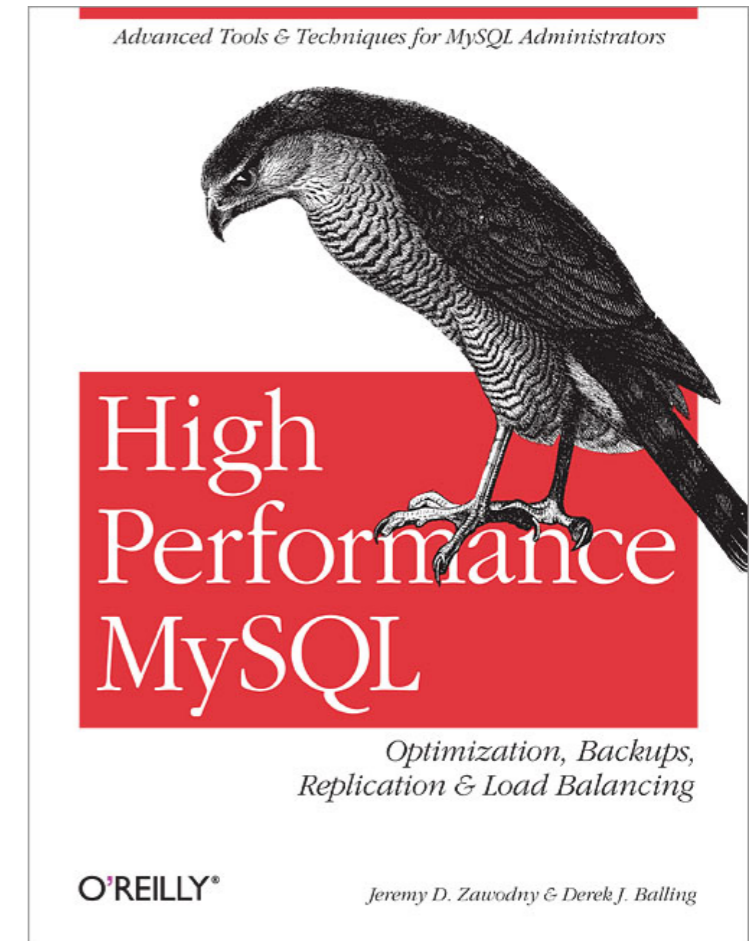
Advice from the Experts



Brad Fitzpatrick,
“Inside LiveJournal’s Backend”



Cal Henderson,
“Building Scalable Websites”



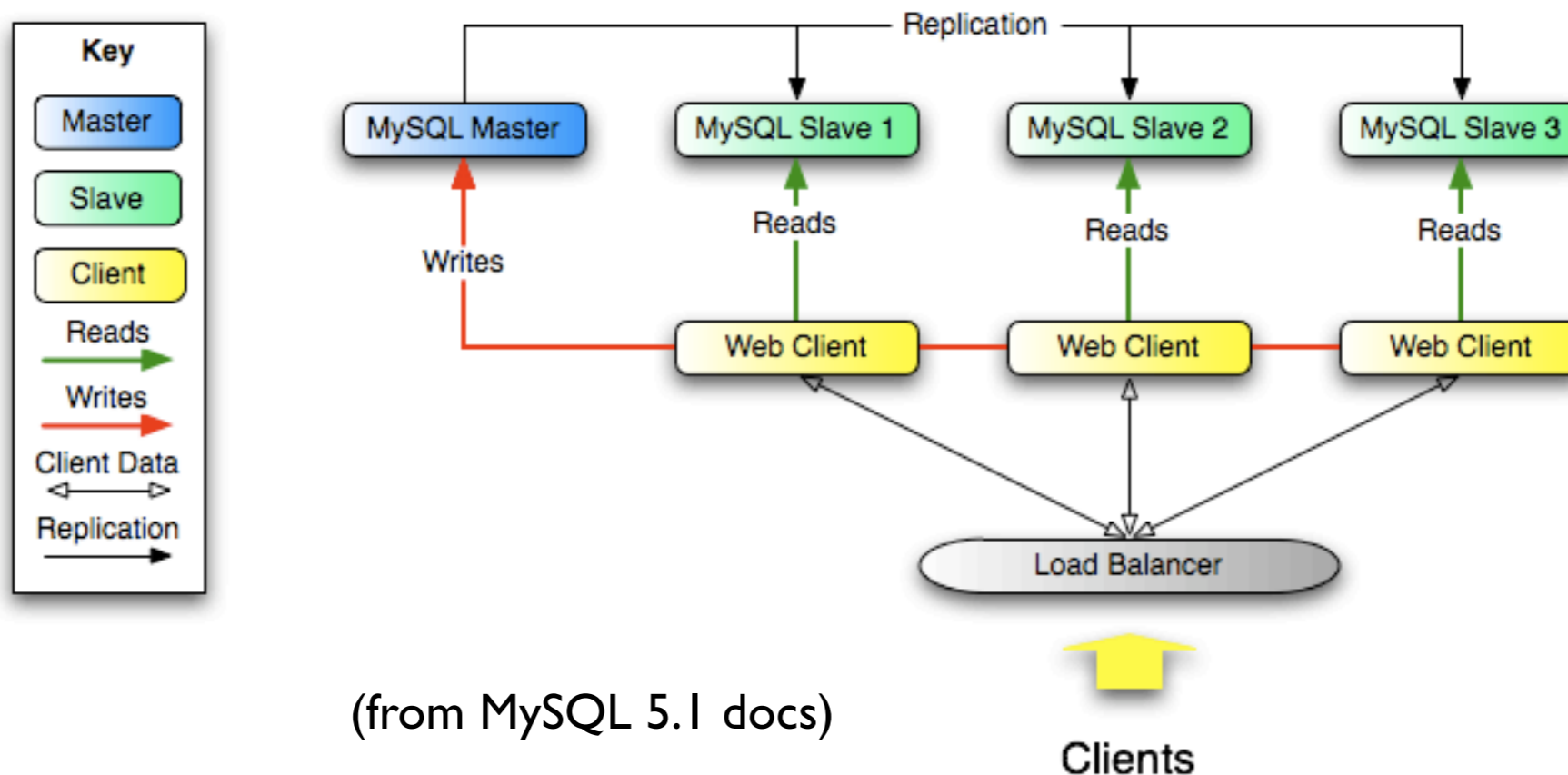
Jeremy Zawodny
and Derek J Balling,
“High Performance MySQL”

Caching

- Keep your hot data in a fast cache
- You get this one for free, thanks to DBI::Gofer
- Can use Cache::FastMmap, memcached, etc. through CHI

Read-only Copies

- Replication to local server or remote slaves
- Be careful of replication lag



Sharding

- Large data is split across multiple machines
 - Users A-L, M-Z
 - Divided by month
 - Consistent hashing algorithm
- May involve directory server
- Bye bye JOINs

- This is going to mean a custom database layer
- And rewriting all your old code to use it
- DBIx::Router tries to separate this plumbing

Shoulders of Giants

- Enter DBI::Gofer
 - “A scalable stateless proxy architecture for DBI”
 - Bundles up requests, sends them over a transport, executes them, sends back results

Shoulders of Giants

- Used by shopzilla.com and petfinder.com to pool connections
- Lots of good stuff like caching, timeouts, tracing
- DBIx::Router is a Gofer transport
 - But it executes the calls locally

Shoulders of Giants

- CPAN makes everything easy
 - SQL::Statement parses SQL (!)
 - Config::Any solves the XML problem

How does it work?

- DataSources
 - Individual (DSNs) or Group
 - Group handles failover
 - Also load-balancing

Single DataSource

```
{  
    name      => 'Master1',  
    dsn       => 'dbi:Pg:dbname=lolcats',  
    user      => 'icanhas',  
    password  => 'ch33zeburger',  
},
```

Group DataSource: random

```
{  
  name      => 'ReadCluster',  
  class     => 'random',  
  datasources => [ 'Slave1', 'Slave2' ],  
},
```

Group DataSource: roundrobin

```
{  
  name      => 'ReadCluster',  
  class     => 'roundrobin',  
  datasources => [ 'Slave1', 'Slave2' ],  
},
```

Group DataSource: roundrobin

```
{  
  name      => 'ReadCluster',  
  class     => 'roundrobin',  
  datasources => [ 'Slave1', 'Slave2' ],  
  failover  => 1,  
  timeout   => 8,  
},
```

Group DataSource: repeater

```
{  
  name      => 'ReadWriteCluster',  
  class     => 'repeater',  
  datasources => [ 'Master1', 'Master2' ],  
},
```

Group DataSource: shard

```
{
  name    => 'StoreShards',
  class   => 'shard',
  type    => 'list',
  table   => 'orders',
  column  => 'store_id',
  shards => [
    {
      values    => [ 1, 3, 5 ],
      datasource => 'EastCoast',
    },
    {
      values    => [ 2, 4, 6 ],
      datasource => 'WestCoast',
    },
  ],
},
```

Subclass DataSourcees

- Custom auth schemes for the paranoid
- Ye olde insane load-balancing scheme
- Shards with a directory server

Rules

- Map queries to DataSources
- Organized in RuleLists
 - Most specific to least
 - Can fall back to pass-through

Rule: regex

```
{
  class      => 'regex',
  datasource => 'ReadCluster',
  match      => ['^ \s* SELECT \b '],
  not_match  => ['\b FOR \s+ UPDATE \b '],
},
```

Rule: readonly

```
{  
  class      => 'readonly',  
  datasource => 'ReadCluster',  
},
```

Rule: parser

```
{  
  class => 'parser',  
  match => [  
    {  
      structure => 'tables',  
      operator  => 'all',  
      tokens    => ['order_history']  
    },  
  ],  
}
```

- Operators: all, any, none, only
- Structures: command, tables, columns

Rule: not

```
{  
  class      => 'not',  
  rule       => { class => 'readonly' }  
  datasource => 'ReadCluster',  
},
```

Rule: default

```
{  
  class      => 'default',  
  datasource => 'Master1',  
},
```

```
{
  datasources => [
    {
      name => 'Master1',
      dsn => 'dbi:mysql:dbname=lolcats',
      user => undef,
      password => undef,
    },
    {
      name => 'Slave1',
      dsn => 'dbi:mysql:dbname=lolcats',
      user => undef,
      password => undef,
    },
    {
      name => 'Slave2',
      dsn => 'dbi:mysql:dbname=lolcats',
      user => undef,
      password => undef,
    },
    {
      name => 'ReadCluster',
      class => 'random',
      datasources => [ 'Master1', 'Slave1', ],
      failover => 1,
      timeout => 8,
    },
  ],
  rules => [
    {
      class => 'readonly',
      datasource => 'ReadCluster',
    },
    {
      class => 'default',
      datasource => 'Master1',
    },
  ],
}
```

How do you run it?

- `DBI_AUTOPROXY="dbi:Gofer:transport=DBIx::Router;conf=/path/to/conf.pl"`
- `$dbh = DBI->connect
("dbi:Gofer:transport=DBIx::Router;conf=/
path/to/conf.pl;dsn=$original_dsn",
$user, $passwd, \%attributes);`

What's the bad news?

- AutoCommit
 - But Tim plans to fix that
- No streaming results
 - Also fixable
- Failover and sharding are tough to generalize

Status

- Hosted on Google Code
- Coming soon to your CPAN mirror
- Mostly there, but shards need work
- Needs the usual docs and tests

Future Directions

- Make routing decisions optionally sticky
- Support explicit hints in method call attributes
 - Helps with sharding
 - Workaround for tricky queries

Thanks!

<http://code.google.com/p/dbix-router/>

What else is out there?

- Mostly faux DBD drivers
 - DBD::Multi
 - DBD::Multiplex
 - DBIx::HA
- DBI::Role