Gofer
A scalable stateless proxy for DBI
Gofer, \textit{logically}

- Gofer is
  - A scalable stateless proxy \textit{architecture} for DBI
  - Transport independent
  - Highly configurable on client and server side
  - Efficient, in CPU time and minimal round-trips
  - Well tested
  - Scalable
  - Cachable
  - Simple and reliable
Gofer, structurally

- Gofer is
  - A simple stateless request/response protocol
  - A DBI proxy driver: DBD::Gofer
  - A request executor module
  - A set of pluggable transport modules
  - An extensible client configuration mechanism

- Development sponsored by Shopzilla.com
Gofer Protocol

- DBI::Gofer::Request & DBI::Gofer::Response
- Simple blessed hashes
  - Request contains all required information to connect and execute the requested methods.
  - Response contains results from methods calls, including result sets (rows of data).
- Serialized and transported by transport modules like DBI::Gofer::Transport::http
Using DBD::Gofer

• Via DSN
  - By adding a prefix
    $dsn = "dbi:Driver:dbname";
    $dsn = "dbi:Gofer:transport=foo;dsn=$dsn";

• Via DBI_AUTOPROXY environment variable
  - Automatically applies to all DBI connect calls
    $ export DBI_AUTOPROXY="dbi:Gofer:transport=foo";
  - No code changes required!
Gofer Transports

- **DBI::Gofer::Transport::null**
  - The ‘null’ transport.
  - Serializes request object, transports it nowhere, then deserializes and passes it to DBI::Gofer::Execute to execute
  - Serializes response object, transports it nowhere, then deserializes and returns it to caller
  - Very useful for testing.

`DBI_AUTOPROXY="dbi:Gofer:transport=null"`
Gofer Transports

- **DBD::Gofer::Transport::stream (ssh)**
  - Can ssh to remote system to self-start server
    
    ```
    ssh -xq user@host.domain \ 
    perl -MDBI::Gofer::Transport::stream \ 
    -e run_stdio_hex
    ```
  - Automatically reconnects if required
  - ssh gives you security and optional compression

```ruby
DBI_AUTOPROXY='dbi:Gofer:transport=stream ;url=ssh:user@host.domain'
```
Gofer Transports

- **DBD::Gofer::Transport::http**
  - Sends requests as http POST requests
  - Server typically Apache mod_perl running
  - DBI::Gofer::Transport::http
  - Very flexible server-side configuration options
  - Can use https for security
  - Can use web techniques for scaling and high-availability. Will support web caching.

```
DBI_AUTOPROXY=’dbi:Gofer:transport=http
;url=http://example.com/gofer’
```
Gofer Transports

- DBD::Gofer::Transport::gearman
  - Distributes requests to a pool of workers
  - Gearman a lightweight distributed job queue
    http://www.danga.com/gearman
  - Gearman is implemented by the same people who wrote memcached, perlbal, mogileFS, & DJabberd

```
DBI_AUTOPROXY='dbi:Gofer:transport=gearman;url=http://example.com/gofer'
```
Pooling via gearman vs http

- I haven’t compared them in use myself yet
  + Gearman may have lower latency
  + Gearman spreads load over multiple machines without need for load-balancer
  + Gearman coalescing may be beneficial
  + Gearman async client may work well with POE
- Gearman clients need to be told about servers

DBD::Gofer

• A proxy driver
• Accumulates details of DBI method calls
• Delays forwarding request for as long as possible
• Aims to be as ‘transparent’ as possible
• Policy mechanism allows fine-grained tuning to trade transparency for speed
• execute_array() is a single round-trip
Three policies supplied: pedantic, classic, and rush. Classic is the default.

Policies are implemented as classes

Currently 22 individual items within a policy

Policy items can be dynamic methods

Policy is selected via DSN:

```
DBI_AUTOPROXY="dbi:Gofer:transport=null
;policy=pedantic"
```
## Round-trips per Policy

```
$dbh = DBI->connect_cached
$dbh->ping
$dbh->quote

$sth = $dbh->prepare
$sth->execute

$sth->{NUM_OF_FIELDS}
$sth->fetchrow_array
$dbh->tables
```

<table>
<thead>
<tr>
<th>pedantic</th>
<th>classic</th>
<th>rush</th>
</tr>
</thead>
<tbody>
<tr>
<td>connect()</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>if not default</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>if not default</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>cached after first</td>
</tr>
</tbody>
</table>
Gofer Caveats

- State-less-ness has implications
  - No transactions. AutoCommit only.
  - Can’t alter $dbh attributes after connect
  - Can’t use temp tables, locks, and other per-connection persistent state, except via stored procedures
  - Code using last_insert_id needs a (simple) change
  - See the docs for a few other very obscure caveats
An Example

Using Gofer for Connection Pooling
The Problem

40 worker processes per server + 150 servers = 1 overloaded database
A Solution

A ‘database proxy’ that does connection pooling

Holds a few connections open

Uses them to service DBI requests

Repeat for all servers
An Implementation

- Standard apache+mod_perl
- DBI + DBD::*
- DBI::Gofer::Execute and DBI::Gofer::Transport::http modules implement stateless proxy
- DBD::Gofer with http transport
Load Balance and Cache

- **server** with 40 workers
- $150 \times \text{servers}$
- $= 6000 \text{ db connections}$
- New middle tier
- **Caching results would reduce db load even further** (not yet implemented)
- **High performance load balancing and fail-over**
- **Far fewer db connections**
Error Handling

- DBD::Gofer can automatically retry on failure
  
  `DBI_AUTOPROXY="dbi:Gofer:transport=null;retry_limit=3"`

- Default behaviour is to retry if
  
  `$request->is_idemponent` is true
  
  - looks at SQL returns true for most SELECTs

- Default is `retry_limit=0`, so disabled

- You can define your own behaviour:

  ```perl
  DBI->connect(..., {
      go_retry_hook => sub { ... },
  });
  ```
# DBD::Proxy vs DBD::Gofer

<table>
<thead>
<tr>
<th>Feature</th>
<th>DBD::Proxy</th>
<th>DBD::Gofer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports transactions</td>
<td>✓</td>
<td>✗ (not yet)</td>
</tr>
<tr>
<td>Supports very large results</td>
<td>✓</td>
<td>✗ (memory)</td>
</tr>
<tr>
<td>Automatic retry on error</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Large test suite</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Minimal round-trips</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Modular &amp; Pluggable classes</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Tunable via Policies</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Scalable</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Connection pooling</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Can support client and web caches</td>
<td>✗</td>
<td>✗ (not yet)</td>
</tr>
</tbody>
</table>
Gofer’s Future

- Caching for http transport
- Optional JSON serialization
- Caching in DBD::Gofer
- Make state-less-ness optional
- Patches welcome!
Future: http caching

- Potential big win
- DBD::Gofer needs to indicate cache-ability
  - via appropriate http headers
- Server side needs to agree
  - and respond with appropriate http headers
- Caching then happens just like for web pages
  - if there’s a web cache between client and server
- Patches welcome!
Future: JSON

- Turns DBI into a web service!
  - Service Oriented Architecture anyone?
- Accessible to anything
  - that can talk JSON
- Clients could be JavaScript, Java, ...
  - and various languages that begin with P or
- Patches welcome!
Future: Client Caching

- DBD::Gofer could access a cache
  - Use serialized request as key to cache
  - If entry found then return that response
- Plug-able caching
  - Would include memcached to give a distributed shared cache
- Patches welcome!
Future: Transactions

• State-less-ness could be made optional
  – If transport layer being used agrees
  – For http that means KeepAlive
  – Easiest to implement for stream / ssh

• Would be enabled by
  AutoCommit => 0
  $dbh->begin_work

• Patches welcome!
Questions?