Boosting The Performance of your PHP Applications

PHP International Conference 2002

November 4, 2002. Frankfurt, Germany

Sterling Hughes & Derick Rethans
 <sterling@bumblebury.com / d.rethans@jdimedia.nl>

Introduction

- PHP optimization
- Server side caching
- Database optimization
- Bandwidth optimization
- External utilities

Reducing Output

Less output is good because...

- Saves server bandwidth
- Decreases server CPU and Memory
usage (well, kinda)
- Has the same effect on the client side
- On large sites, bandwidth is most
often the most costly bottleneck

Optimize

- Use preprocessors to eliminate
whitespace
- If you save 20 bytes per file, and have 1,000,000 hits a day, you've just saved 20,000,000 extra bytes
- Send content "when ready", and use progressive images

HTML Optimization

- Remove All Comments!
- Meta tags don't work!
- Use parent HTML tags to specify child values, when all children have the same value
- Fully qualify links (ie, personal/images/
not personal/images)

CSS Optimization

- Use CSS instead of incessant <font> tag usage
- All CSS should be contained in an external file that can be cached.
- Use short names for CSS attributes
- Override global HTML tag attributes using CSS properties
- Use CSS to control table definitions with CSS v2...

```
<STYLE TYPE="text/css">
<--
   TABLE {table-layout:fixed; width: 100%}
   #fixed COL {width: 50%}
-->
</STYLE>

<TABLE>
   <COL><COL>
   <TR><TD>Cell 1</TD><TD>Cell 2</TD></TR>
   <TR><TD>Cell 3</TD><TD>Cell 3</TD></TR>
</TABLE>
```

URL Shortening

By using Apache's mod_rewrite module, you can greatly shorten your HTML files by making common locations into one/two letter aliases.

Apache Mod_Rewrite Entry

```
RewriteMap      abbr            dbm:/etc/apache/abbr
RewriteRule     ^/r/([^/])/?(.)   ${abbr:$1}$2
[redirect=permanent,last]
```

/etc/apache/abbr

```
b    bytes/
d    dogs/
g    graphics/

gt    graphics/template/
```

HTTP Caching

Use HTTP headers to have finer grained control over browser caching, these headers include:

Headers

```
Last-Modified: Mon, 22 Jul 2002 15:50:00 GMT
Expires: Mon, 22 Jul 2002 15:52:25 GMT
Cache-Control: must-revalidate, max-age=15, s-maxage=0, private
```

Compress

HTTP supports native compression by using Lempel-Ziv encoding (LZ77). Its the same encoding mechanism used by the popular unix tool <code>gzip</code>.

Compression Advantages

- Greatly reduces bandwidth
- Cross browser compliant
- Compression can cache as well

Compression Disadvantages

- CPU intensive on the Server
- CPU intensive on the Client

PHP supports output compression via PHP's ouput buffering features and the zlib extension.

The simplest way is to use the <code>ob_start ('ob_gzhandler')</code> or use the configuration options <code>zlib.output_compression</code> and <code>zlib.compression_level</code>

HTTP_Compress

Alternatively, you can use PEAR's excellent HTTP_Compress class that will handle everything related to HTTP_Compression for you...

```
<?php
    require_once 'HTTP/Compress.php';

    HTTP_Compress::start ();
    print "Some data\n";
    print "Some more data\n";
    print "Even more data\n";
    HTTP_Compress::output ();
? >
```

mod_gzip

Mod_gzip is a superb apache module which handles gzip compression and data caching. It offers signifigant increases in compression performance to the homebrewed PHP solutions.

mod_gzip features

- Compression statistics
- The ability to chose compression via MIME type
 and Filename
- You can exclude compression for certain browsers
- Can cache compression results

Server side caching

- Caching dynamic content
- Pregenerating content
- Creating pages on demand

Do not use PHP!

PHP...

   ...leads to slower pages.

Slower pages...

   ...lead to fewer users.

Fewer users...

   ...lead to less income.

Static rulez!

How dynamic is your content?

Do use PHP!

but

- Use as much static content as possible
- Cache dynamic generated content
- Use as few database queries as possible

- Simple caching
- Caches pages, blocks, functions

PEAR::Cache_Lite - Example

```
<?php
    / Include the class /
    require_once('Cache/Lite.php');

    / Set a key for this cache item /
    $id = 'newsitem1';

    / Set a few options /
    $options = array(
        'cacheDir' => '/var/cache/news/',
        'lifeTime' => 3600
    );

    / Create a Cache_Lite object /
    $Cache_Lite = new Cache_Lite($options);

    / Test if there is a valid cache-entry for this key /
    if ($data = $Cache_Lite->get($id)) {
        / Cache hit! /
    } else {
        / Cache miss! /
        ob_start();
        / Generate content /
        $data = ob_get_contents();
        $Cache_Lite->save($data);
    }
? >
```

Caching: Pros and Cons

Pros:

- Less use of external resources

Cons:

- PHP is still parsing the page
- Cache expiry time
- Stale content

Pregenerated content

- Only regenerate page when needed
- Different elements

Pregenerating: Pros and Cons

Pros:

- No stale content
- PHP is only used for regeneration

Cons:

- Costs a lot of resources
- Hard to figure out what to regenerate

Creating pages on demand

- Only regenerate page when requested
- Most pages are never requested
- Use the 404-trick

On demand - example

Facts:

Url: http://yoursite.com/news/000001.html

Path: /home/httpd/news/

Set up a 404 error handler:

```
<Directory /home/httpd/news>
    ErrorDocument 404 /generate-news.php
</Directory>
```

generate-news.php:

```
<?php
    $id = basename($_SERVER['REDIRECT_URL'], '.html');

    / Generate the HTML /
    include "/home/httpd/news/generator.php";
    $html = generate_html($data);

    / Show the page /
    echo $html;

    / Store the page for later use /
    $fp = fopen(sprintf("/home/httpd/news/%06d.html", $id), "w");
    fputs($fp, $html);
    fclose($fp);
? >
```

On demand: Pros and Cons

Pros:

- No wasted resources on generating
- No stale content

Cons:

- None :-)

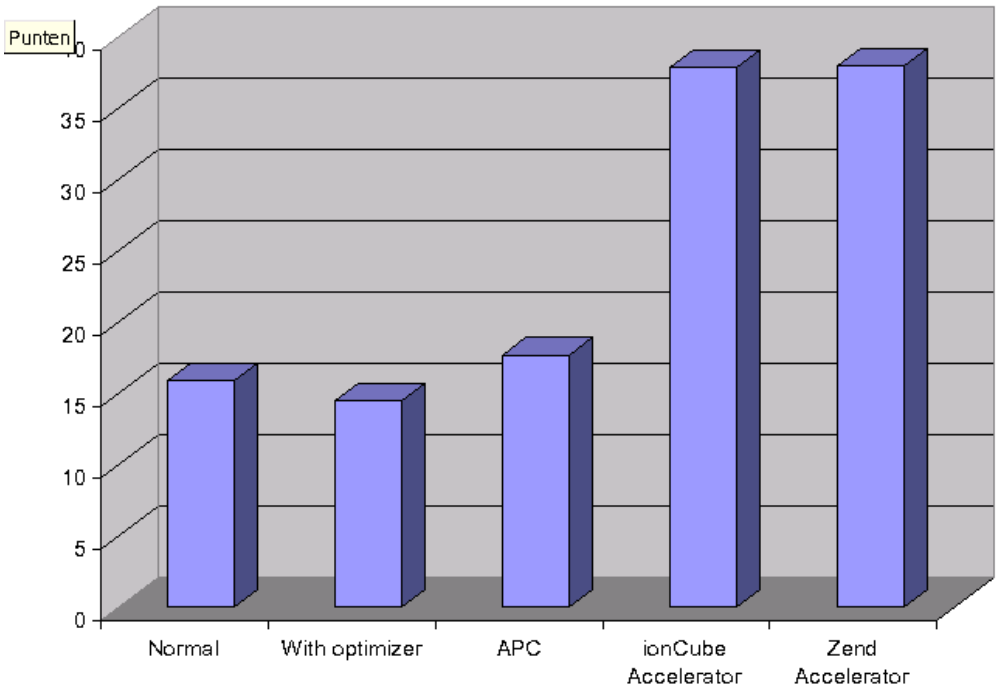
External Tools

- Bytecode caches
- Content caches
- SRM: Script Running Machine
- lingerd

Bytecode caches

- Cache compiled PHP scripts
- Serializing caches:
- APC (Alternative PHP Cache)
- 'SHM' caches:
- ionCube Accelerator
- Zend Cache

Cache Benchmark



- transparent caching
- caches HTTP and FTP

Some effort to use with PHP:

```
<?php
$tsstring = gmdate("D, d M Y H:i:s ", $timestamp)."GMT";

if ($_SERVER["HTTP_IF_MODIFIED_SINCE"] == $tsstring) {
    / The UA has the exact same page we have. /
    header("HTTP/1.1 304 Not Modified");
    exit();
} else {
    header("Last-Modified: ".$tsstring);
}
? >
```

SRM: Script Running Machine

- Functions to work with persistent Objects
- Bridge between PHP Client and Objects
- Objects run in threads
- Manage persistent resources and data

- Lingering conections
- Apache hands over the job
- Apache can do useful things
- Less server load

What is lingerd?

Lingerd is a daemon (service) designed to take over the job of properly closing network connections from an http server like Apache.

Because of some technical complications in the way TCP/IP and HTTP work, each Apache process currently wastes a lot of time "lingering" on client connections, after the page has been generated and sent. Lingerd takes over this job, leaving the Apache process immediately free to handle a new connection. As a result, Lingerd makes it possible to serve the same load using considerably fewer Apache processes. This translates into a reduced load on the server.

Lingerd is particularily useful in Apache webservers that generate dynamic pages (e.g in conjunction with mod_perl, mod_php or Java/Jakarta/Tomcat).

More importantely, lingerd can only do an effective job if HTTP Keep-Alives are turned off; since keep-alives are useful for images, the recommended lingerd setup is to have an Apache/mod_whatever/lingerd server for the dynamic pages, and a plain Apache (or thttpd or boa) for the images.

Performance

So You Want to get performance from your php applications?

Performance Problems

- Makes you network dependent (TCP/IP)
- Communication protocol overheads (TCP/IP, especially)

Performance Tips

- When dealing with local sockets consider other
 options like pipes or shared memory.
- Establishing a connection is the most costly
 part of the TCP/IP layer. Think carefully about re-using pre-existing connections.
- When transfering large chunks of data, consider using a compression/encryption method to lower bandwidth

File I/O Problems

- Disk access is slower than memory access (duh!)
- Simultaeneous write/write operations cause data corruption. Therefore locking must occur.
- Locking is slow

File I/O Tips

- Use the disk in concentrated bits, meaning do all
your writing together.
- Do Not Read Large Files Into Memory!
- Do Not Read Small Files From Disk!
- Employ caching to access frequently used data
 stored in memory
- Try and put write operations where they will
 cause the least loss in user experience (remember,
 web pages are chunked!)

- Avoid Using frequent array accesses, remember, each array access is a hash lookup.

The Wrong Way

```
<?php
$ar['name'] = 'Sterling';

for ($i = 0; $i < 20; ++$i) {
    print $ar['name'];
}
? >
```

- Shorten "required" executions, such as loops.
- Avoid rebuilding large data structures.
- Look for alternatives to regular expressions

Slow Regexps

Its often beneficial to replace regular expressions with simpler functions that have the same effect.

With Regular Expression

```
<?php
if (preg_match ('/(.)@(.)/', $data, $matches)) {
    print $matches[2];
}
? >
```

Without Regular Expressions

```
<?php
$pos = strrpos ('@', $data);
if ($pos !== false) {
    print substr ($data, $pos);
}
? >
```

Fast Regexps

In other cases it may help to use regular expressions instead of custom parse routines in your PHP code.

SQL Problems

- Unused data is being pulled over a socket
- FULLTEXT indexes
- the LIKE clause

SQL Tips

- Study and Learn Database Optimization
- Failing that use the following tips...
- MySQL supports the 'LIMIT' clause which can be used
 to lower the returned result set to a specified number
- Never use LIKE or a FULLTEXT index to do complex
 searches. Use keywords and an alternative ranking system
- The only thing worse than using a LIKE clause, is building
 your own LIKE clause in PHP.

Database optimization

- Queries are slow
- Recursive queries are slower
- Recursive queries on every page are slowest

Visitation model 1

Directory structure:

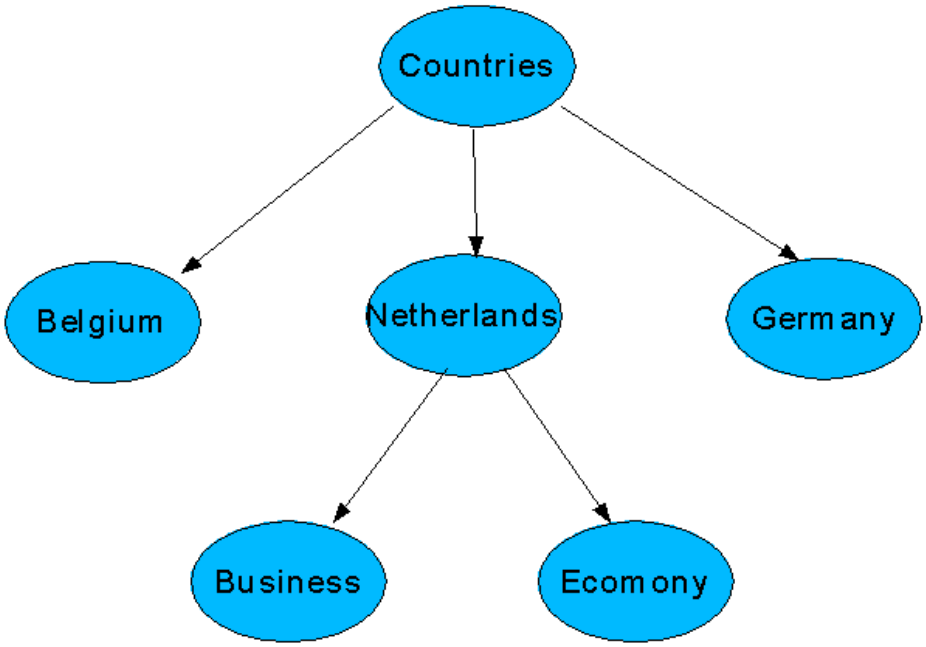
Stored as: <table class="db" border="1" align="center" cellspacing="0"> <tr> <th>Id</th><th>Parent</th><th>Name</th> </tr> <tr> <td>3</td><td>1</td><td class="left">Countries</td> </tr> <tr> <td>4</td><td>3</td><td class="left">Belgium</td> </tr> <tr> <td>5</td><td>3</td><td class="left">Netherlands</td> </tr> <tr> <td>6</td><td>3</td><td class="left">Germany</td> </tr> <tr> <td>7</td><td>5</td><td class="left">Business and Economy</td> </tr> </table>

Retrieve list with:

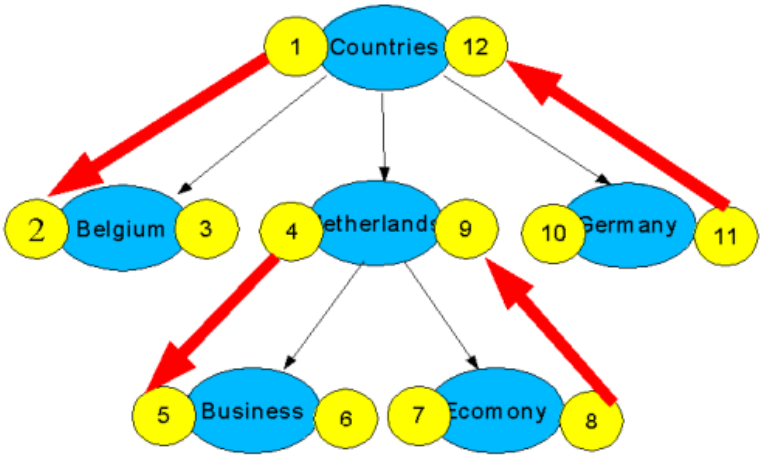
```
<?php
    function gimmeParent($id) {
        global $dir;

        $res = mysql_query("SELECT parent, name FROM directory WHERE id = $id");
        if (mysql_num_rows($res) > 0) {
            $dir[] = ($row = mysql_fetch_row($res));
            gimmeParent($row['parent']);
        }
    }

    gimmeParent(7);
? >
```

Visitation model 2

Tree structure:



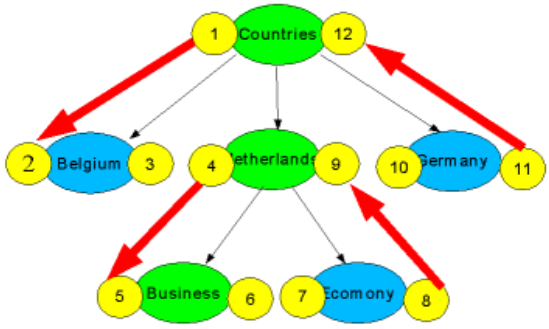
Visitation:



Stored as: <table class="db" border="1" align="center" cellspacing="0"> <tr> <th>Id</th><th>Parent</th><th>Name</th><th>Left</th><th>Right</th> </tr> <tr> <td>3</td><td>1</td><td class="left">Countries</td><td>1</td><td>12</td> </tr> <tr> <td>4</td><td>3</td><td class="left">Belgium</td><td>2</td><td>3</td> </tr> <tr> <td>5</td><td>3</td><td class="left">Netherlands</td><td>4</td><td>9</td> </tr> <tr> <td>6</td><td>3</td><td class="left">Germany</td><td>10</td><td>11</td> </tr> <tr> <td>7</td><td>5</td><td class="left">Business</td><td>5</td><td>6</td> </tr> <tr> <td>8</td><td>5</td><td class="left">Economy</td><td>7</td><td>8</td> </tr> </table>

Visitation model 3

Select the path:

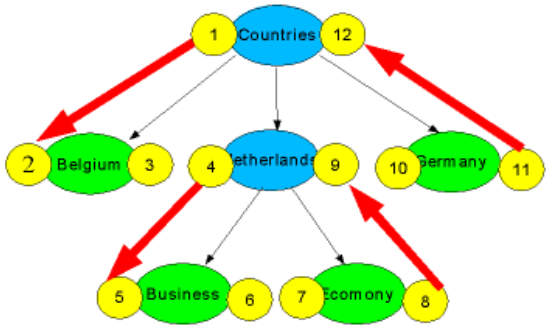


Query:

```
SELECT * FROM directory
WHERE
    left < 5 AND right > 6
```

Visitation model 4

Select all end points:

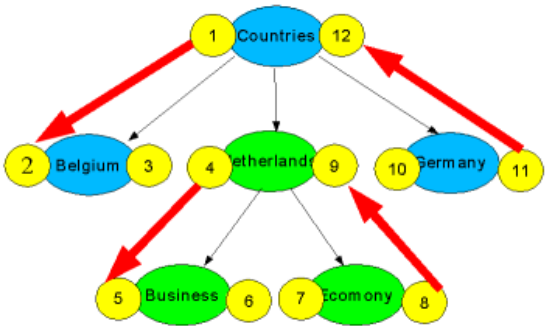


Query:

```
SELECT * FROM directory
WHERE
    right - left = 1
```

Visitation model 5

Select a subtree:



Query:

```
SELECT * FROM directory
WHERE
     left > 4 AND right < 9
```

Thank You!

These Slides: http://pres.derickrethans.nl

HTTP Cache Explanation: http://www.mnot.net/cache_docs/

HTML Tidy: http://www.w3.org/People/Raggett/tidy/

ionCube: http://www.ioncube.com

lingerd: http://www.iagora.com/about/software/lingerd

mod_gzip: http://www.remotecommunications.com/apache/mod_gzip/

PHP: http://www.php.net

squid: http://www.squid-cache.org

Webreference HTML Optimization: http://www.webreference.com/authoring/languages/html/optimize/

Zend: http://www.zend.com

Index

Introduction 2
Reducing Output 3
Optimize 4
HTML Optimization 5
CSS Optimization 6
HTTP Optimization 7
Compress 8
PHP Compression 9
mod_gzip 10
Server side caching 11
Do not use PHP! 12
Static rulez! 13
Do use PHP! 14
PEAR::Cache_Lite 15
PEAR::Cache_Lite - Example 16
Caching: Pros and Cons 17
Pregenerated content 18
Pregenerating: Pros and Cons 19
Creating pages on demand 20
On demand - example 21
On demand: Pros and Cons 22
External Tools 23
Bytecode caches 24
Cache Benchmark 25
Squid 26
SRM: Script Running Machine 27
lingerd 28
Performance 29
Sockets 30
File I/O 31
CPU Usage 32
Regular Expressions 33
SQL Queries 34
Database optimization 35
Visitation model 1 36
Visitation model 2 37
Visitation model 3 39
Visitation model 4 40
Visitation model 5 41
Thank You! 42