Full Stack Web Application Performance Tuning

codecentric GmbH
About the presenter

Fabian Lange, 27
married, no children but 2 cats

- Fabian & symfony
  Fabian.Lange@symfony-project.com
  Using symfony since late 2006
  Core team member since early 2008
  Release manager for symfony 1.2

- Fabian & work
  Working as Java & Web architecture and performance consultant at codecentric
  www.codecentric.de
  Previously worked for Nokia / Nokia Siemens Networks and Hewlett Packard

- Fabian & blogging
  Company blog at blog.codecentric.de
  Private blog at blog.hma-info.de
About codecentric

- **Performance**
  Specialized in Java Performance optimization, codecentric has 100% success rate in finding and solving performance issues at a huge number of customers. We strive to continue this rate using state of the art tools and processes.

- **Architecture**
  Consulting software architects or managers, assisting with the design and creation of modern software architecture, weather it shall be a SOA architecture or classical design, our experts will get you there.

- **Open Technology**
  Cost efficient Open Source Tools and Solutions complete our portfolio. We believe that using Open Technologies improve software products quality and are more cost efficient than reinventing the wheel.

  - [www.codecentric.de](http://www.codecentric.de)
    See our full portfolio, meet the team and get our contact details.
Scope of this presentation

- Make web applications feel faster
  Ideally a stopwatch can prove out effort
  Performance is perceived differently
  Tweak at the end of development cycle

- Not an expert guide on specific issues
  But see the whole picture
  No micro-tweaking

- Targeted at small to medium deployments
  Larger ones have dedicated people or high end hardware
  Application developer tweaks single server
  Keep scaling options open – share nothing
What is the Whole Picture?

- **Client Software**
  - Browser

- **Web Framework**
  - symfony

- **Server Software**
  - PHP
  - Web Server
  - Database

- **Server Hardware**
  - Processor
  - Memory
  - Storage
Client Software – Browsers

- Rendering priority
  - Javascripts at bottom
    - Can be done by explicitly outputting them in the template
      ```php
      echo include_javascripts();
      ```
    - To make this possible avoid inline Javascript or use Event.observe
    - Watch out with using JS before it is loaded
      - For example mouseover events
      - Prototype Event.observe as well!

  - CSS at top
    - Layout can be already calculated when building the DOM tree.

  - Image dimensions
    - Same as with CSS. Browser knows the image size before it loads the image
      Not only for performance, but flickering UI looks bad as well.
Client Software – Browsers

- Obtaining HTML is often the smallest part of the browser latency

- Limited connections (2-6) additionally slow down loading
  
  Use a separate host for assets
  
  You can even use another server for your media files
  
  Even better try consider using a CDN -> less load
  
  Cookies are not sent to other domains, which reduces request size

- Get rid of unneeded additional requests
  
  Many pages use a lot of images
  
  CSS Sprites

  ```
  #nav li a {background-image:url('/img/image_nav.gif')}
  #nav li a.item1 {background-position:0px 0px}
  #nav li a.item2 {background-position:0px -32px}
  ```

  Inline Image Data (RFC2397)

  ```
  <img src="data:image/gif;base64, ASCIIDATA" />
  ```
Client Software – Browsers

- Get rid of unneeded additional requests (2)

  Combine JS & CSS
  - Automation tools available
  - Sensible modularization allows separate update cycles and selective inclusion
    - Protoculous: 1 file, but pages that just want Prototype have to parse Scriptaculous

  Minify JS and CSS
  - Easy and compatible using YUI Compressor
  - Dean Edwards Packer minifies better but produces runtime overhead

  Use expires for assets
  - Note: You need to change file name for users to see the change
    - ExpiresActive On
    - ExpiresDefault "access plus 10 years"

  Use redirects wisely
  - Redirects make sense to prevent double postings
  - Redirects are responses to the browser with a instruction to load a new page
Performance Tweaks

ETags

used to save bandwidth, but does not help with latency
calculates a hash code of the returned content and only returns content if
doesn’t match browsers Etag
doesn’t work well with external compression

Core Compilation

symfony creates a single PHP files with all its core classes
Comments are removed from this file
Own classes can be added via app/config/core_compile.yml

- %SF_ROOT_DIR%/apps/frontend/lib/myUser.class.php
- %SF_ROOT_DIR%/lib/AutoLinker.class.php

Should be done only with classes
### Propel

Use Propel 1.3 instead of 1.2 as it is faster and less memory consuming

- Mainly benefits from object instance pooling

Monitor “slow queries” log

Use `EXPLAIN $slow_query`

Add index where no index used

```sql
SELECT section.ID, section.TITLE, section.TREE_LEFT, section.TREE_RIGHT, section.TREE_PARENT, section.TREE_SCOPE FROM `section` WHERE section.TITLE='news' LIMIT 1;
```

<table>
<thead>
<tr>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMPLE</td>
<td>section</td>
<td>ALL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>2121</td>
<td>Using where</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMPLE</td>
<td>section</td>
<td>const</td>
<td>title</td>
<td>title</td>
<td>768</td>
<td>const</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Use `Peer:doSelectJoinXXX` when many related objects are loaded

Reduce unneeded hydration

```php
setUserID($comment->getUserId()) instead of setUser($comment->getUserId())
```

Reduce unneeded queries

DB access is the most expensive part, it is always good idea to use few queries.

In extreme cases consider tweaking the database model by moving columns and merging tables, as a perfectly normalized database model is slow due to many joins.
Caching

Caching is very easy!

- Create a cache.yml in the config directory of a module
- List the actions/components/partials you want to cache
  - `action:`
    - `enabled: on`

Two important options:
- `with_layout`
- `contextual`

There are more cacheable partials than expected

Caching nearly always improves response times

Don’t build complex cleaning

- Introduces risk of not cleaning everything
- Clean every hour
- Clean more than needed
  - New wildcard cleaning in symfony 1.1
### symfony is modular

It is easy to bypass parts

```php
#ajax_frontController.php

require_once(dirname(__FILE__).'/../config/ProjectConfiguration.class.php');
$config = ProjectConfiguration::getApplicationConfiguration('frontend', 'prod', false);
$context = sfContext::createInstance($config);

$action = $context->getController()->getAction('ajax','myMethod');
$action->initialize($context);
$action->executeMyMethod();

$context->getResponse()->sendHttpHeaders();
$context->getResponse()->sendContent();
```

It is easy to exchange parts

```yaml
#app/config/factories.yml

all:
  storage:
    class: myMemorySessionStorage
```
Session Storage

PHP stores user sessions by default in files.
symfony does the same by using sfSessionStorage.
Files on the filesystem can be slow, can be a security risk and are hard to scale.
symfony ships alternative Session Storage classes.

```
#app/config/factories.yml
storage:
    class: sfPDOSessionStorage
    param:
        database: pdo  # Database connection to use
        db_table: sessions  # Name of table storing the sessions
        db_id_col: id  # Name of column storing the session id
        db_data_col: data  # Name of column storing the session data
        db_time_col: timestamp  # Name of column storing the session time
```

Note: reusing a propel connection for MySQL does not work.
Server Software

- Server Software often differs, especially in versions

- PHP
  
  PHP itself gains performance boosts with every release -> later is better
  

- Web Server
  
  Both Apache HTTPD and LightHTTPD work fine
  

  Using PHP via FastCGI brings huge performance boost!

  But brings also troubles, e.g. for APC user cache

- Database
  
  A standard setup usually ships a MySQL

  InnoDB should be preferred because of its tuning options and reliability

  MyISAM is claimed to be “faster” but in fact that is rarely the case.

  Of course other DBs are good as well 😊
Server Software – PHP

- As PHP becomes faster with each version, there should be no real reason to stay with old versions
- PHP itself allows not much performance tweaking
- A PHP opcode compiler should be mandatory, and actually will be included in PHP 6 by default
  
  **APC**
  ```
  extension=apc.so
  apc.stat=0
  Disables APC checking for PHP file modifications
  Requires a server restart to detect updated files
  ```

  **xCache**
  ```
  extension = xcache.so
  ```
Server Software – Apache 2.2

- **Multi Processing Module**
  - Prefork
    - Will launch Apache processes that launch multiple PHP instances
  - Worker
    - Will launch Apache processes that launch one PHP instance with multiple threads
    - Worker works fine with most symfony applications, but some modules might not work

- **Logging**
  - Frequently accessed resources will cause excessive logging.
  - To selectively turn off logging use this:
    ```
    #apache2.conf
    CustomLog /var/log/apache2/access.log combined env=!dontlog
    #Site/v-host config
    SetEnvIf Request_URI "^/chat" dontlog
    ```
  - Or perhaps turn off logging at all 😊
Server Software – Apache 2.2

- Compression
  There are 3 methods of compression possible:
  - Let Apache compress
  - Let PHP compress
  - Let symfony compress
  Apache compression is most proven and easy to configure
  AddOutputFilterByType DEFLATE text/html text/plain
text/xml text/css application/x-javascript
  works also on non PHP & non symfony content
  Make sure to only compress with one option

- Other Tweaks
  HostnameLookups off
  will prevent reverse lookup of hostname for IPs
  ExtendedStatus off
  will prevent apache collecting too much status information
Keep alive

This HTTP 1.1 feature works roughly like this:

- When a browser connects to load a page, a connection is opened.
- Instead of closing the connection after returning, let’s say the HTML, with keep-alive the connection is kept open.
- When one of two criteria is reached, the connection is closed. This can be either a number of requests, or a timeout.

The upside of this feature is that some connection overhead is reduced.
The downside is that resources are wasted in idle periods.

Often the browser will load images directly after the HTML, unless they are cached.

Many guides recommend turning this feature off, but for small to medium loaded servers, this can improve the responsiveness.

A balanced config could look like:

- `KeepAlive On`
- `MaxKeepAliveRequests 1000`
- `KeepAliveTimeout 3`

Symfony 1.0 - Need to patch symfony response class:

```php
sfWebResponse::sendHttpHeaders();
```

HTTP/1.0 → HTTP/1.1
Server Software – MySQL 5

- MySQL configuration done in config file
  
  `/etc/mysql/my.cnf`

MySQL itself has multiple sample configurations, check them

- Generic MySQL settings
  
  `key_buffer = 32M`
  
  MyISAM but also for temp tables

  `thread_cache_size = 32`
  
  Creating threads is slow, try to get “threads created” low

  `table_cache = 1K`
  
  Opening tables is slow

  `sort_buffer_size = 16M`
  
  Used for sorting results, increase if “sort_merge_passes” increases

  `query_cache_limit = 1M`
  
  Caches how queries are executed, does not bring much but also does not cost much

  `query_cache_size = 32M`
  
  Caches results of queries, but too high settings involve high maintenance costs
Server Software – MySQL 5 & InnoDB

- **Disable Logging**
  
  ```
  # log = ...
  ```

- **Specify Slow Logging for Optimisation**
  
  ```
  log_slow_queries  = /var/log/mysql/mysql-slow.log
  long_query_time   = 2
  log-queries-not-using-indexes
  ```

- **InnoDB settings**
  
  ```
  innodb_log_file_size  = 250M
  
  Data written goes into here first
  
  innodb_buffer_pool_size = 1G
  
  Main InnoDB data cache. Waste memory here!
  
  innodb_additional_mem_pool_size = 20M
  
  Utility buffer for InnoDB
  
  innodb_thread_concurrency  = 8
  
  innodb_flush_log_at_trx_commit  = 2
  
  2 will flush to OS cache, 1 will write to disc every commit
Server Hardware

- **Processors**
  More are better, but scales only until I/O wait
  Even the most basic server offerings have a dual core by now

- **Memory**
  Reduces I/O wait, as data can be in memory
  An Apache process running symfony needs up to 64 MB
  Also useful for database caching
  Think of: DB Size x 2

- **Storage**
  Web servers mostly spend time on loading database, html or asset data
  Get fast discs. Noise doesn’t matter in server rooms
  Use at least a two disc setup, if no raid

- **Main hardware issue is I/O wait!**
Server Hardware – Two Disc Setup

- Even without expensive and perhaps even complicated raid configurations a second harddisc might significantly boost the server performance

- Separate discs by responsibility

- Example:
  One user does a heavy DB operation
  In meantime 10 other users want to browse the page, requesting 10 images each
  One disc can read/write for the DB operation continuously
  The other disc can handle the many small read bursts for images

- Simple config for InnoDB:
  ```
  innodb_log_group_home_dir = /var/disc1/log/mysql/iblogs
  innodb_data_home_dir = /var/disc2/mysql
  innodb_data_file_path = ibdata:2000M:autoextend
  ```
References

- Symfony
  - http://www.symphony-project.org/book/1_1/18-Performance
  - http://trac.symphony-project.org/wiki/Optimization

- Server:
  - http://httpd.apache.org/docs/2.2/misc/perf-tuning.html
  - http://www.devside.net/articles/apache-performance-tuning
  - http://www.mysqlperformanceblog.com

- CSS & JavaScript
  - http://www.alistapart.com/articles/sprites
codecentric GmbH
Grünewalder Str. 29-31
42657 Solingen

phone    +49-212-2494315
fax      +49-212-2494109
email    info@codecentric.de