My site is slow

Hernâni Borges de Freitas
Technical Consultant
hernani@acquia.com
@hernanibf

Madrid, 20th October, 2012

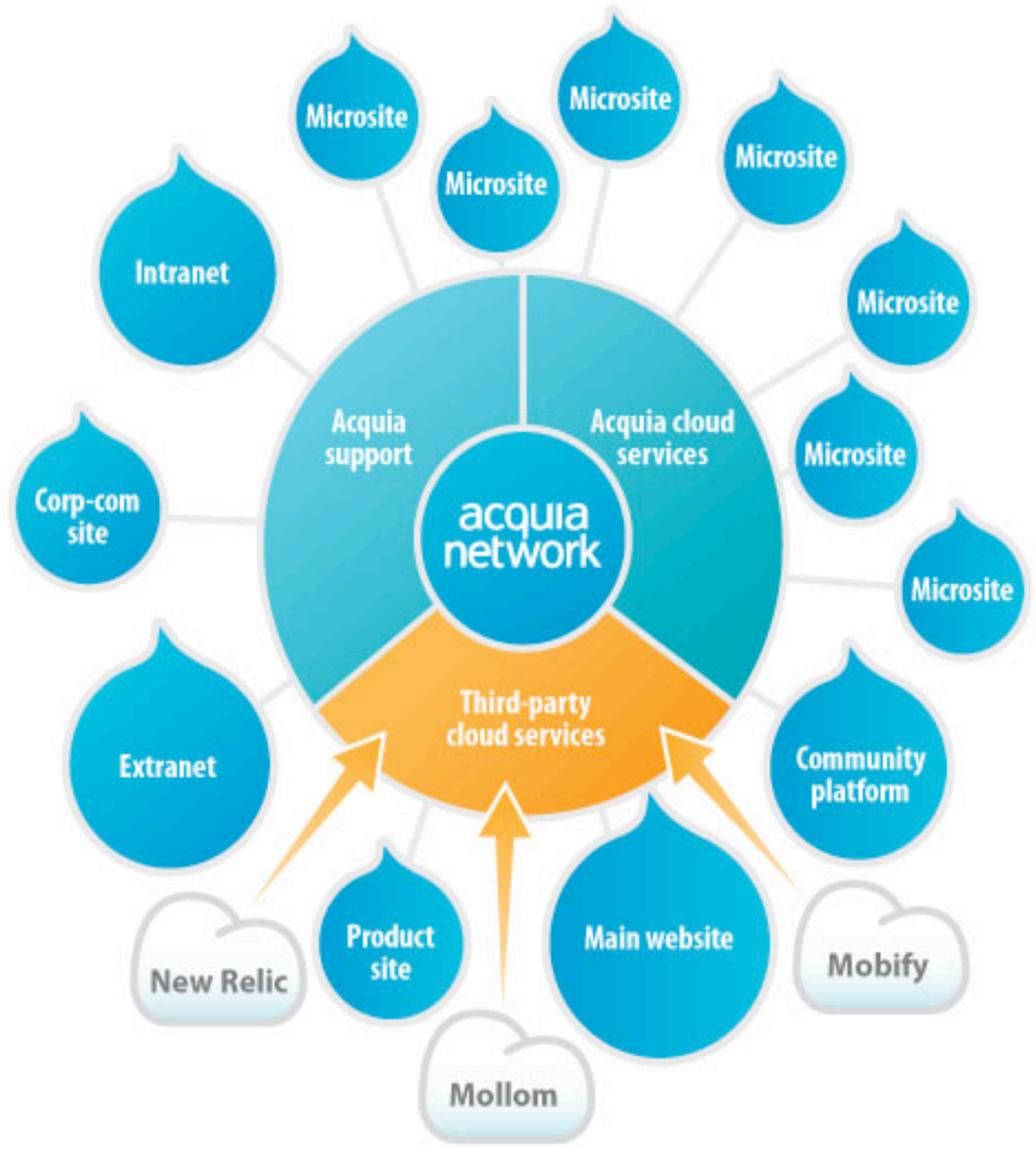
Acquia™

About me

- .PT
- Acquia Professional Services EMEA
- Technical Consultant
- Drupal* many things
- Travel lover
- Twitter.com/hernanibf

About us

- Expert Drupal Support
- Optimized Drupal hosting
  - Dev Cloud
  - Managed Cloud
- Foster Drupal adoption
  - Commons
  - Drupalgardens.com
  - Dev Desktop



What does my team do ?

- Drupal Jumpstarts
- Architecture Workshops
- Discovery workshops
- Site Audits

- **Performance Assessment**
- Security Audits
- On-site Consulting

This is how it starts

"My site is slow."

Acquia™

This is what usually follows.

"We already added 5 more webservers..."

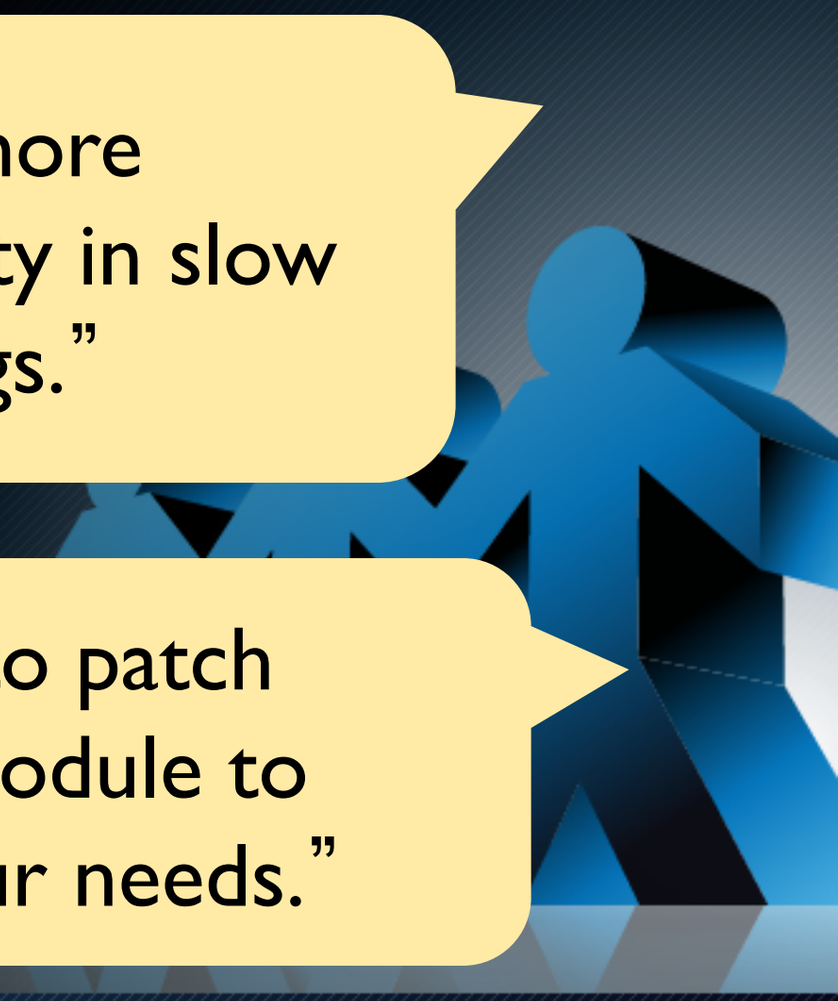
"We are thinking about adding more mysql slaves…"

"We added nginx, changed MySql Version, added memcache..."

"We need more granularity in slow query logs."

"We are sure the problem is in PHP Version."

"We tried to patch boost module to serve our needs."



But sometimes it's even worst.

"We need to rebuild this site from scratch".

"We need to uber-cache the whole site."

"We will export the site to pure HTML and ditch the CMS."

"We can't handle this traffic with Drupal".

"It's time to move to node.js"

The main problem ?

Every site is different

- Act without analyzing, thinking and measuring.
- Before touching your application / server stack, you need:
  - Data about current performance of production system.
  - Data about what you are doing

What you mean? “It’s slow”?

- **Backend** slowness
  - Services that website use are slow or unresponsive (dbs)
  - Application too complex
  - Server resources overload
- **Frontend** slowness
  - Too many assets
  - Slow connection between browser and server.
  - JS slowing the DOM (re)rendering

Always true

"If your site is slow, before installing boost, varnish, last caching module goodie, last crazy technique you read in a blog, **you need to stop and install your brain.**".

Specially true for sites with:

- Authenticated users.
- Access Control.
- Lots of complexity and interactions.

Science

Base your work in what you can prove

- Find good data that you can use to **measure**.
- Find good profiling data you can use to **analyze**.
- **Implement** improvements for the problems found.
- **Measure** again.

Data about your site

Understanding how much time are you taking per page

You need data to understand your current situation and compare after your changes.

- Apache logs with rendering times

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
```

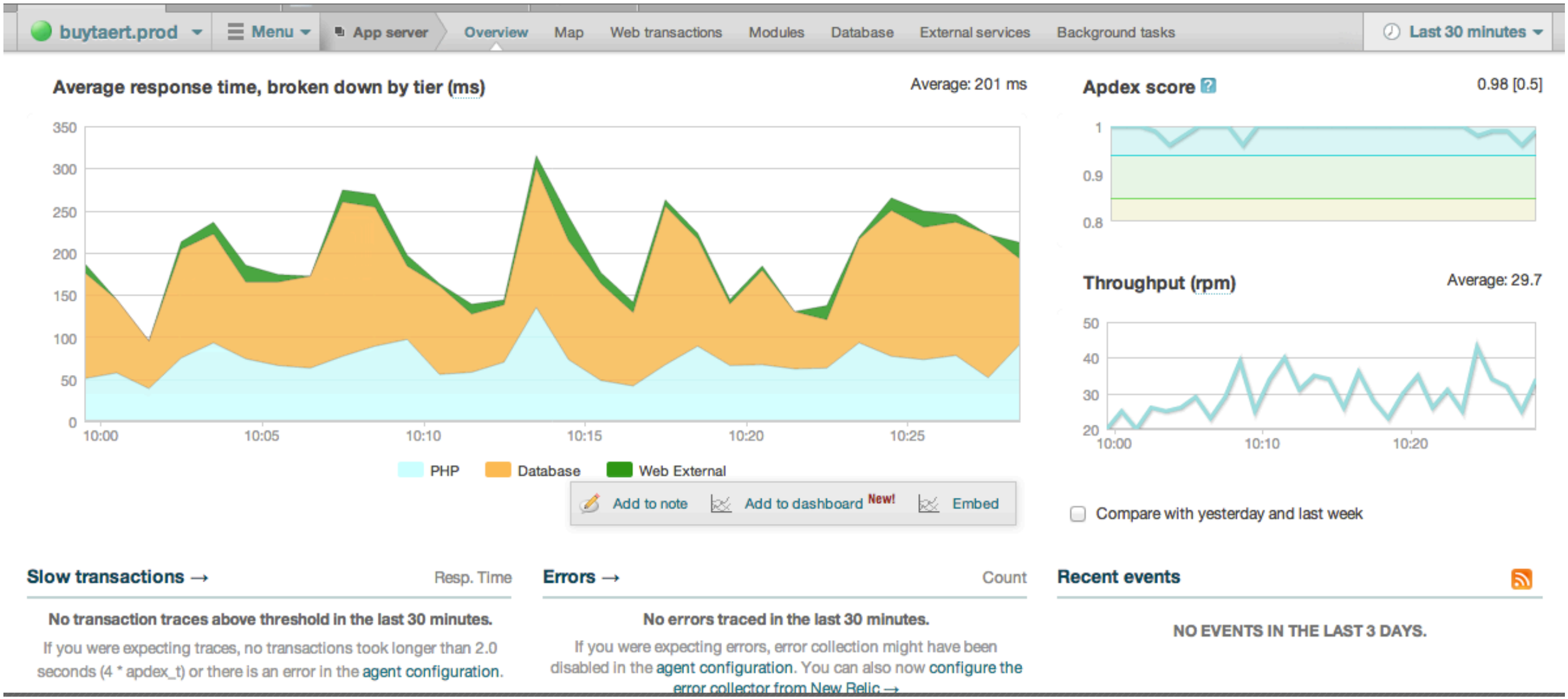
```
[31/Jan/2008:14:19:07 +0000] "GET / HTTP/1.1" 200 7918 ’" 0/95491
```

- Accesslog from statistics table

Aid	Title	Path	Ip	Uid	Timer	Timestamp
23557145	Directory	node/465	10.156.11.24	2632	7262806	1346855808
23557361		home	10.156.11.24	9263	6967768	1346856897
23557399		home	10.156.11.24	7904	6690128	1346856985
23557081	Search	search/node/h	10.156.11.24	8058	6509745	1346855538

NewRelic.com

Extra insight about how are you doing

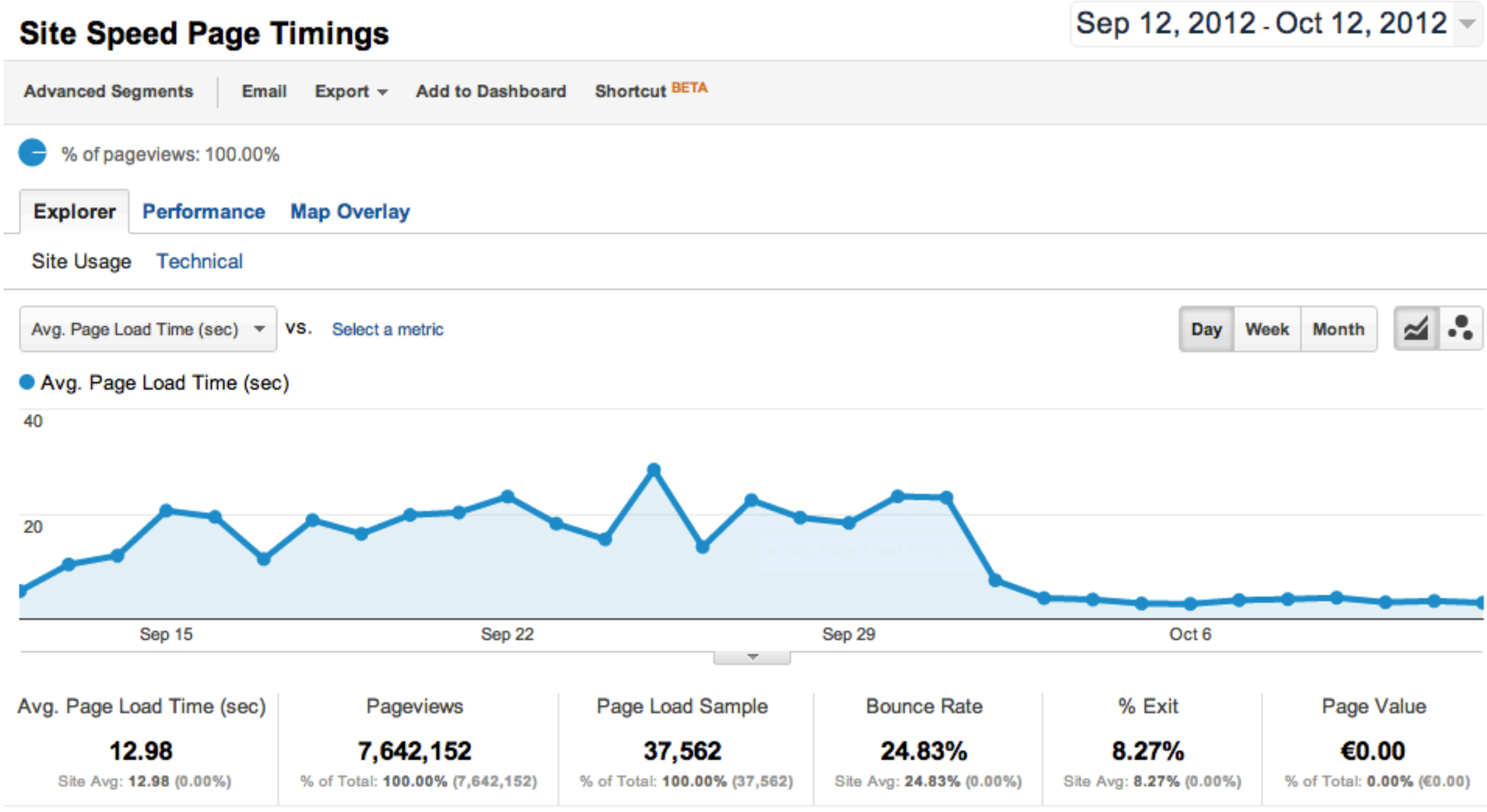


My site is slow

Acquia™

Google Analytics site speed

Be sure you know how fast you are delivering front end



My site is slow

Acquia™

Analyze on a deeper level

Generate your own data

- **XhProf**
- **XhProfCli**

Aggregates data from XhProf profiling data from a set of pages.

https://github.com/msonnabaum/XHProfCLI

Overall Summary	
Total Incl. Wall Time (microsec):	56,326,712 microsecs
Total Incl. CPU (microsecs):	18,709,171 microsecs
Total Incl. MemUse (bytes):	1,682,424,024 bytes
Total Incl. PeakMemUse (bytes):	1,771,511,368 bytes
Number of Function Calls:	3,024,945

What you need to find

Before you do anything else

- Find average/individual page load times.
- Find average/individual page memory consumption.
- Identify problematic pages.
- Compare time wasted in CPU vs Waiting for IO.
- Identify problematic components (blocks, views, components).

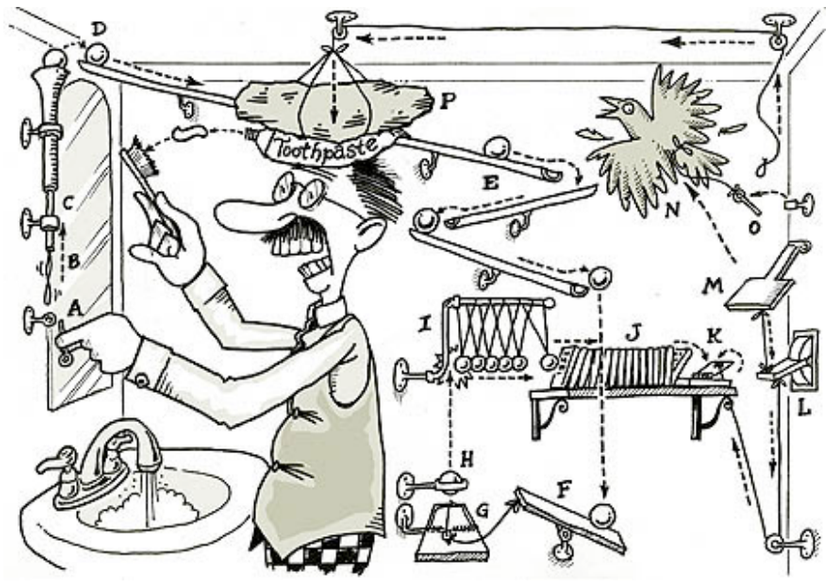
Profile

Time for some research

Look for pages you suspect

- Start by easy ones
  - 404 page (the fastest page you can get).
  - Node view page
  - Homepage
- Continue with the ones your data marked as slow.

Benchmarks



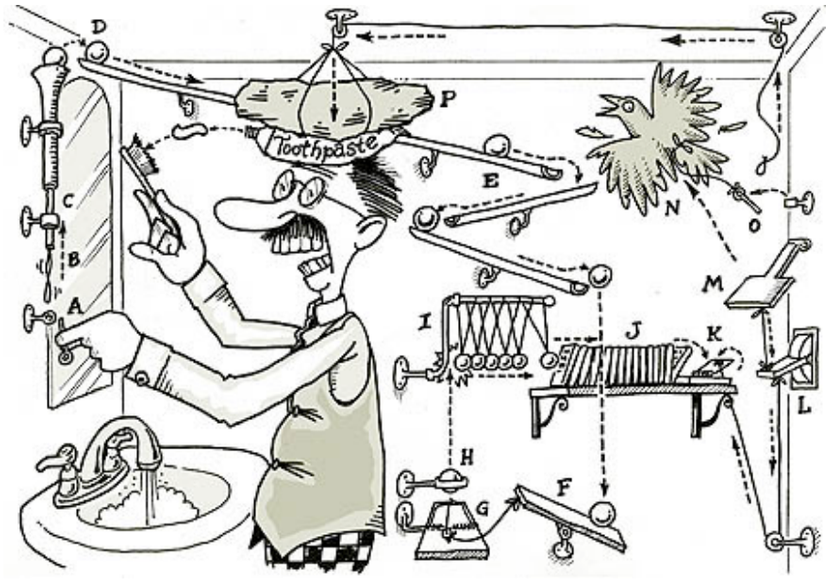
Ideally your normal pages should take

- 1 ~ 1.5 sec
- 40 ~ 60 mb of memory
- 100~300 queries per page

Simpler pages like 404 are good indicators of what is the fastest all other pages will run.

Measure pages for normal users, not for super admin without access checks and the heavy admin_menu in all pages...

Profiling tools



Chasing it

- Use **Devel** module (http://drupal.org/project/devel) to have a fast indication of page load times and memory consumption.
- Use **XhProf** to profile the page and understand slower components.
- Use **timer_start()**, **timer_read()** functions in situations where you are unsure.

Short Demo

Typical #1 – The single SLOOOOW Query

My site is slow

Acquia™

Typical #1 – The single SLOOOOW Query

- First look to profiling data shows something really slow.

Devel

Executed 104 queries in 4900.72 ms. Queries exceeding 5 ms are **highlighted**. Page execution time was 6700.92 ms. Memory used at: devel_boot()=**0.82** MB, devel_shutdown()=**56.6** MB, PHP peak=**50.7** MB.

XhProf

Overall Summary	
Total Incl. Wall Time (microsec):	6,745,032 microsecs
Total Incl. CPU (microsecs):	1468,029 microsecs
Total Incl. MemUse (bytes):	56,558,616 bytes
Total Incl. PeakMemUse (bytes):	57,051,720 bytes
Number of Function Calls:	69,180

Problem

- Related to the database (Wall time vs Total Time).
- Number of queries is low, so probably it's a single query.
- Unfortunately, occurs in a specially visited page.
- Unfortunately, they are not usually single.

Typical #1 – The single SLOOOOW Query

- Not indexed queries, views query with a lot of nodes,

```
SELECT node.nid AS nid, users.picture AS users_picture, users.uid AS users_uid, users.name AS
users_name, users.mail AS users_mail, node.title AS node_title, GREATEST(node.changed,
node_comment_statistics.last_comment_timestamp) AS node_comment_statistics_last_updated
FROM node node
        INNER JOIN users users ON node.uid = users.uid
        INNER JOIN node_comment_statistics node_comment_statistics ON node.nid =
node_comment_statistics.nid
ORDER BY node_comment_statistics_last_updated DESC
```

- Count(*) queries by views pagers with a lot of nodes in InnoDb.

```
4000.71 1    views_plugin_pager::execute_count_query   P A E    SELECT COUNT(*) AS expression FRO
                                                               field_data_field_location ON node
                                                               :views_join_condition_0 AND field
                                                               taxonomy_term_data_field_data_fie
                                                               taxonomy_term_data_field_data_fie
                                                               (:db_condition_placeholder_1)) ))
```

Typical #2 – The super fast queries

Typical #2 – The super fast queries

Devel

Executed 640 queries in 3375.62 ms. Queries exceeding 5 ms are **highlighted**. Page execution time was 6235.46 ms. Memory used at: devel_boot()=**0.82** MB, devel_shutdown()=**80.12** MB, PHP peak=**83.75** MB.

XhProf

Overall Summary	
Total Incl. Wall Time (microsec):	6,745,032 microsecs
Total Incl. CPU (microsecs):	1468,029 microsecs
Total Incl. MemUse (bytes):	80,558,616 bytes
Total Incl. PeakMemUse (bytes):	83,051,720 bytes
Number of Function Calls:	120,180

Problem

- High number of queries
- High memory consumption
- High number of function calls
- All those little queries and memory consumption mean that you are loading lots of information from the database.

Typical #2 – The super fast queries

Usually

- Big Menus with lots of menu items.
- Page contains elements that are slow to render (several views, minipanels in megamenu, nodes loaded everywhere).
- Nodes/users/menus being loaded all the time:

```
Search-result.tpl.php
<?php
  $nid = $result[node]->nid;
  $node = node_load($nid);
```

Typical #3 – The castaway

Typical #3 – The castaway

- Edge conditions that occur in every page load

```
// we will need to mark users who saw the xmas page
function xmas_init() {
        global $user;
        $user = add_xmas_tree_to_user($user);
        user_save($user);
        variable_set('xmas_is_coming', 1);
}
```

```
// connect to webservice to get information and show in footer homepage
function hook_footer() {
        $webservice = new WeatherService();
        $weather_c = $webservice->get('weather', 'Madrid');
        if ($_GET['q'] == 'home')
          return t('%degrees degrees.', array('%degrees', $weather_c));
}
```

Typical #3 – The castaway

- More common things on this topic
  - Blocks rendered. But not shown.
  - Menus computed to a special task. But not used.
  - Theme_rebuild and cache_clear_all in middle of code. Yes code executed in frontend pages.

Typical #4 – The missile killer

Typical #4 – The missile killer

- Usually a task executed in special situations or in certain pages that seriously slows down the platform.
  - Synchronizations of thousand of nodes from web services.
  - Synchronization of all user base from LDAP.
  - Sending thousand of mails via Cron.
  - Saving a node, clearing the full cache.
  - Even worst when those tasks are called by frontend pages.

Typical #5 – Killing me softly

- Several small percentages take as much hit as a big one.

theme_closure	29	0.0%	10,161,929	18.0%
view::render	51	0.0%	10,124,902	18.0%
googleanalytics_footer	29	0.0%	10,122,479	18.0%
token_replace_multiple	145	0.0%	9,984,977	17.7%
token_get_values	435	0.0%	9,860,922	17.5%
menu_execute_active_handler	30	0.0%	9,500,425	16.9%
views_block	43	0.0%	9,104,335	16.2%

- It's almost 18% of all page load times to simply render Google Analytics Js.
- Look to sections or functions that take more than 10% of time and look if they can be improved.

Solutions

After you identified the problems

1. Reduce complexity. Make sure your site is as slim as possible.
2. Cache where you can. At all levels.
3. Maintain cache as longer as possible until it is acceptable.
4. Compute behind the scenes when you can.
5. Distribute the heavier tasks to larger intervals.
6. Grow infrastructure if you are reaching server limits.

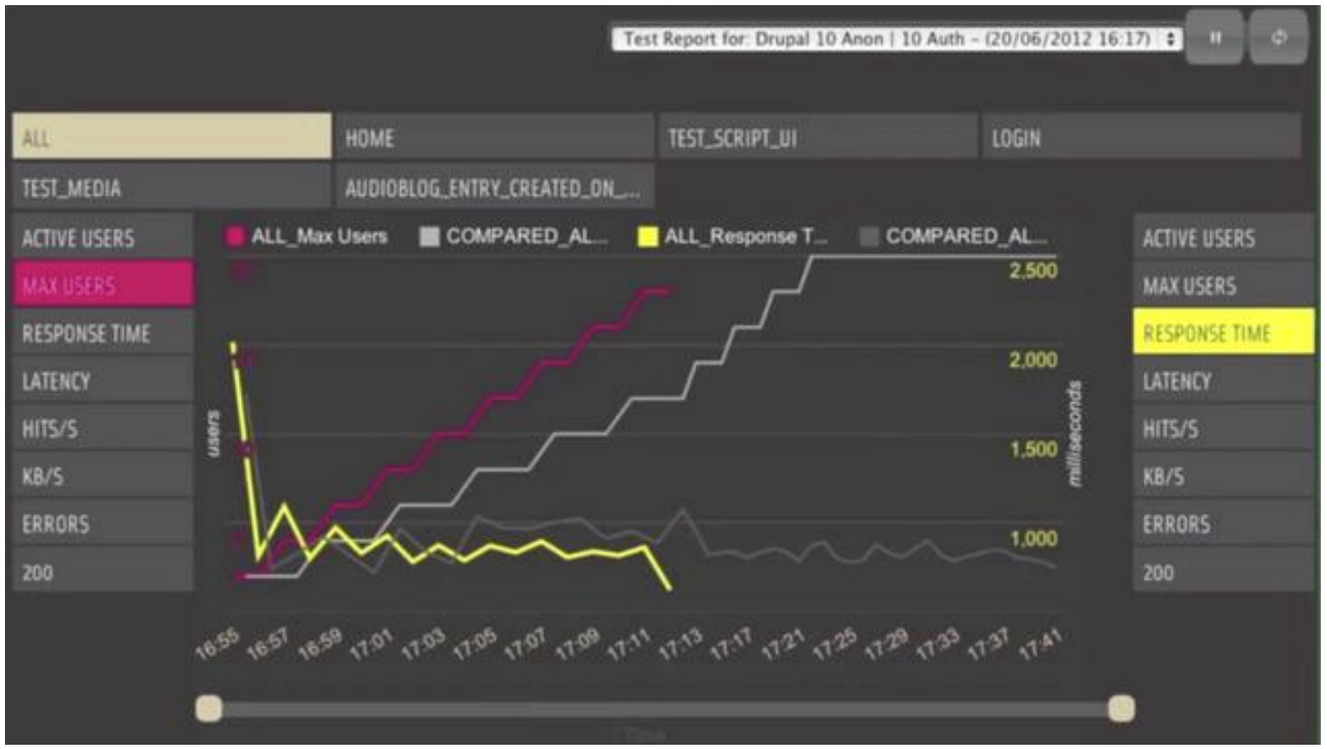
Performance

Can it be cached? Cache it!

- Page caching, block caching, panels caching, views caching, caching API..
- Review caching strategy:
  - https://www.acquia.com/blog/when-and-how-caching-can-save-your-site-part-2-authenticated-users
- Guarantee that caching is effectively helping you.
  - Don't clear it too often.
  - Not used only by a minority.

Measuring in high load

- With sample load
  - Locally Jmeter with a list of urls
- With decent load
  - Blazemeter
  - Blitz.io
- Measure again a subset of results.



So, before your questions.
I do have a question.

Would you like to join Acquia?

We are hiring EVERYWHERE!

- Consultants
- Support
- Sales
- Engineering

Twitter.com/hernanibf

QUESTIONS ?

Acquia™