

# php+mysql and scalability

examples from yahoo, facebook, and canadahomehealth

1

## who i am

- ali asaria
- developed all the software and implementation for canadahomehealth.com
- computer engineering, uwaterloo -- but i am not a web-expert -- self-taught

2

# my first traffic scaling problem

- uwstudent.org got slashdotted
  - (article about uw selling it's soul to microsoft)
- PHP + MySQL
- what to do? only one guy knew enough webserver trickery to static-afy the hot page
  - problem solved; i realize i know nothing

3

# my next scaling problem

- canadahomehealth.com had been running for 6 months
- get an email one day from shared hosting plan server company:
  - “your site is making too many SQL queries. you have 2 days to leave”
  - AAAAAAAAAHHHHHH!

4

# what is scalability?

- weird people on the net argue about this
- an exact definition only becomes important when people start saying stupid things
- a general notion:
  - how well can your site/app scale to more traffic/users
  - scalable software is software that will scale to larger problems with simple hardware changes

5

# php & scalability

- people argue about this a lot, too
- php is, in theory, inefficient. it isn't pre-compiled, it is clunky and loosely defined, messy, "i have a function for everything"-type language (remember visual basic?)
- but... facebook uses php+mysql. haha, take that!

6

# canadahomehealth: implementation

- CanadaHomeHealth.com
  - online health and beauty store
  - written in PHP + MySQL using lots of open-source pieces patched together + my own code: osCommerce, zenCart, WordPress.
  - ~500 products at launch, was soon 1000... 2000...
- running on LAMP
- our server rides the short-bus to school (but we love her anyway)

7

## classic scalability problem

- you've got a dynamic site, traffic is already at 80% capacity, what do you do?

8

# some options

- install a php speed-er-upper
- static-afy
- more hardware,
  - separate mysql from apache, have distributed mysql server slaves. distributed apache.
- dns distributor/reverse proxy
- re-write your software, redesign your site

9

# my solution: what i couldn't do

- can't use a standard php cache-er (e.g. jpcache) because ALL our pages are dynamic (shopping cart, session ID, user ID)
- can't add hardware. why? hardware = \$, startup = poor. but also, i don't know how! (ali = not that smart)
- reverse dns/squid not possible -- needs extra hardware and even MORE smarts

10

# my solution

step-by-step

11

## step 1: redesign the site

- strip out unnecessary dynamic content.
- find out what parts of the site were making queries and ask if you need them
- net reduction:
  - queries = 50% fewer
  - page load = 30% faster.

12



13

```
9 for (...)
10 {
11     printTitle($postID);
12     printContent($postID);
13     printAuthor($postID);
14     ...
15 }
16

19 for (...)
20 {
21     $post->getDetails($postID);
22     $post->printTitle();
23     $post->printContent();
24     $post->Author();
25 }
```

14

# step 3: intelligently cache

- brute, last-step caching (e.g. jpcache) won't work for us -- will actually be slower.
- but so much of my site changes infrequently!
- break it up and cache tiny modules, use output buffering and filesystem
- net savings:
  - 1/100th of the queries (!)
  - 2-3X faster

15



16



17

code stripped out

2

1 code stripped out

3

18



# step 4: php interpreter cache

- php is an *interpreted language*, so a text-file is converted to code on every read. but the text .php file hardly ever changes (never changes?)
- wouldn't it be better to convert to code once, and only re-interpret when the text .php file changes?
- enter ionCube accelerator.
  - free. one file. 30 seconds to install
- net savings: 30% faster, same # of queries



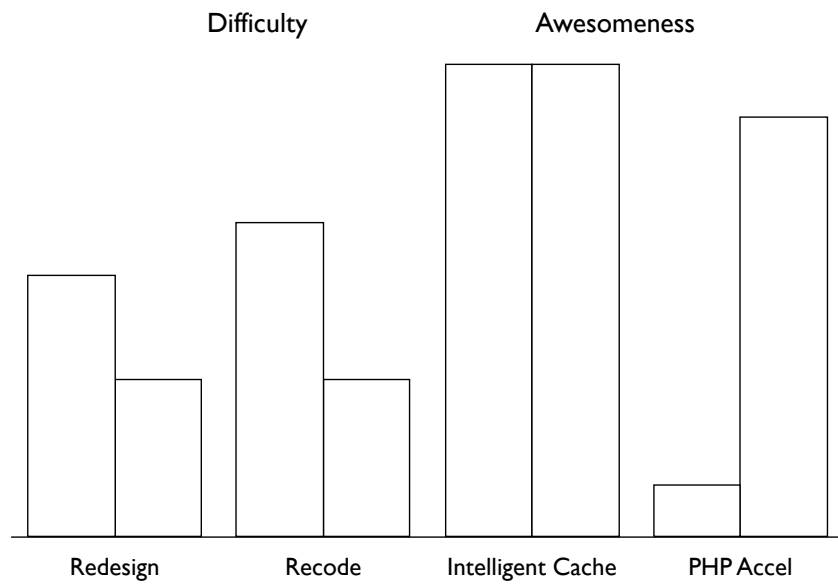
21

## summary of savings:

30 times faster on high load  
300+ queries per page -> 4 or 5

22

# Comparison of Methods



23

## Other Options

(if you're smarter and/or richer)

24

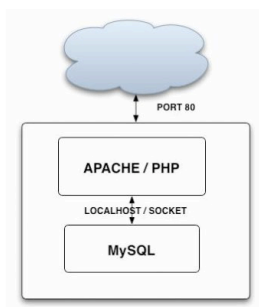
# faster machine

- better processor, more processors
- more memory
- faster disk
- other schnazzle

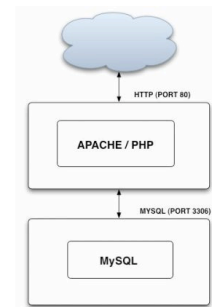
25

# separate mysql box

ONE BOX



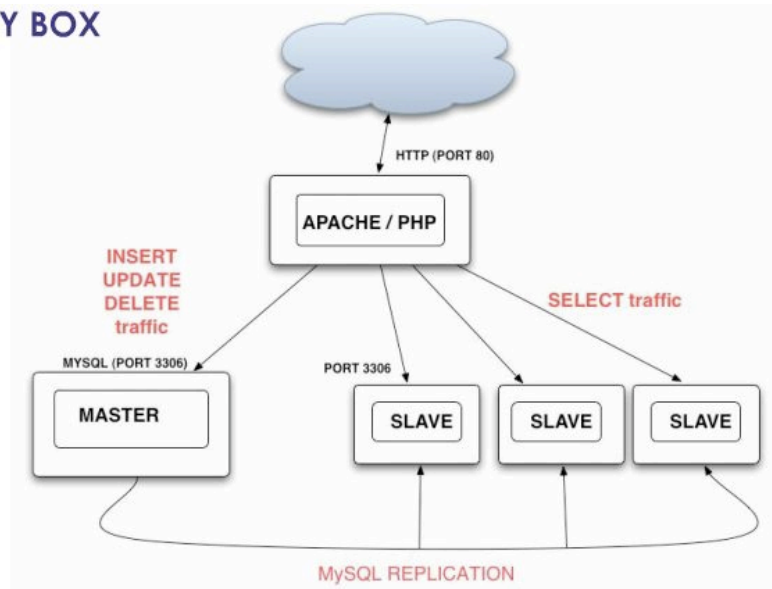
TWO BOX



26

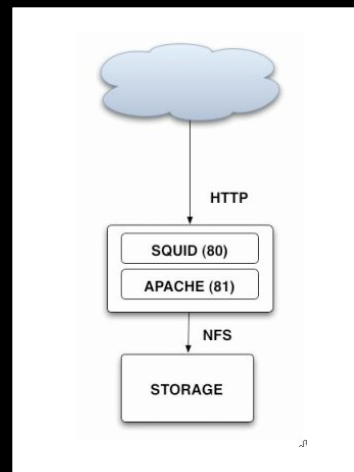
# database replication

## MANY BOX



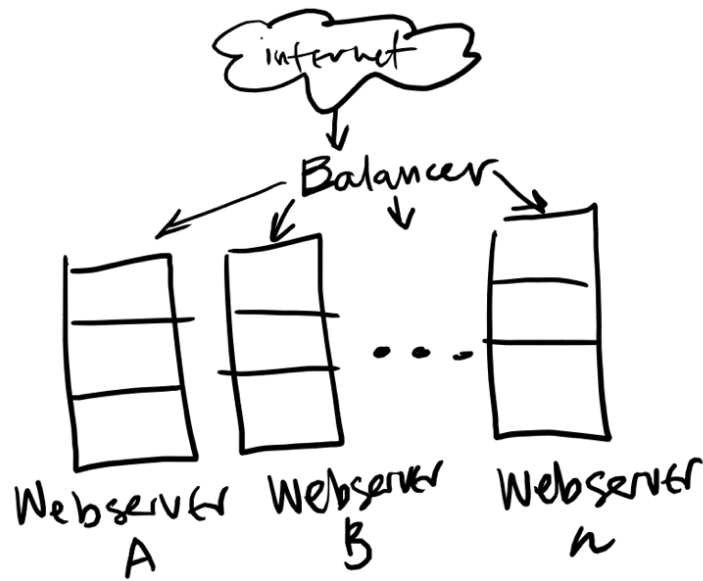
70

27



70

28



## load balancer

29

divide up your users  
before the load balancer

- uwaterloo.facebook.com,  
harvard.facebook.com, etc...
- google.com, google.ca, etc.
- www1.site.com, www2.site.com, ...
- blog.site.com, wiki.site.com
- negatives: bad for SEO, inelegant

30

# crazy memory cache

- facebook uses memcached
- the idea: store objects in a huge (e.g. 64 GB) sea of memory. check the memcache before looking things up.
- super fast

31

## one last interesting note

have we realized how much caching is going on?

32

# layers of caching that happen when you visit a facebook page

- user's browser cache (can have +3 sub-levels)
- squid cache
- photo cache
- load balancer (not a cache)
- memcached
- php interpreter cache
- smart cache
- mysql's built-in caches

33

## summary

- use a multilayered caching approach, plan ahead
- smart code-level caching makes the biggest diff
- ionCube accelerator is wicked good
- php might suck, but still...
- possible to make a well-coded site more than 10 times faster/efficient without any changes to hardware
- eventually you're going to have improve hardware, at which point you need someone who knows what she/he is doing...

34

end.

ali asaria

[ali@canadahomehealth.com](mailto:ali@canadahomehealth.com)