softonic
Let's download!

"How to pimp high volume PHP websites"

27. September 2008, PHP conference Barcelona

By Jens Bierkandt

Jens Bierkandt

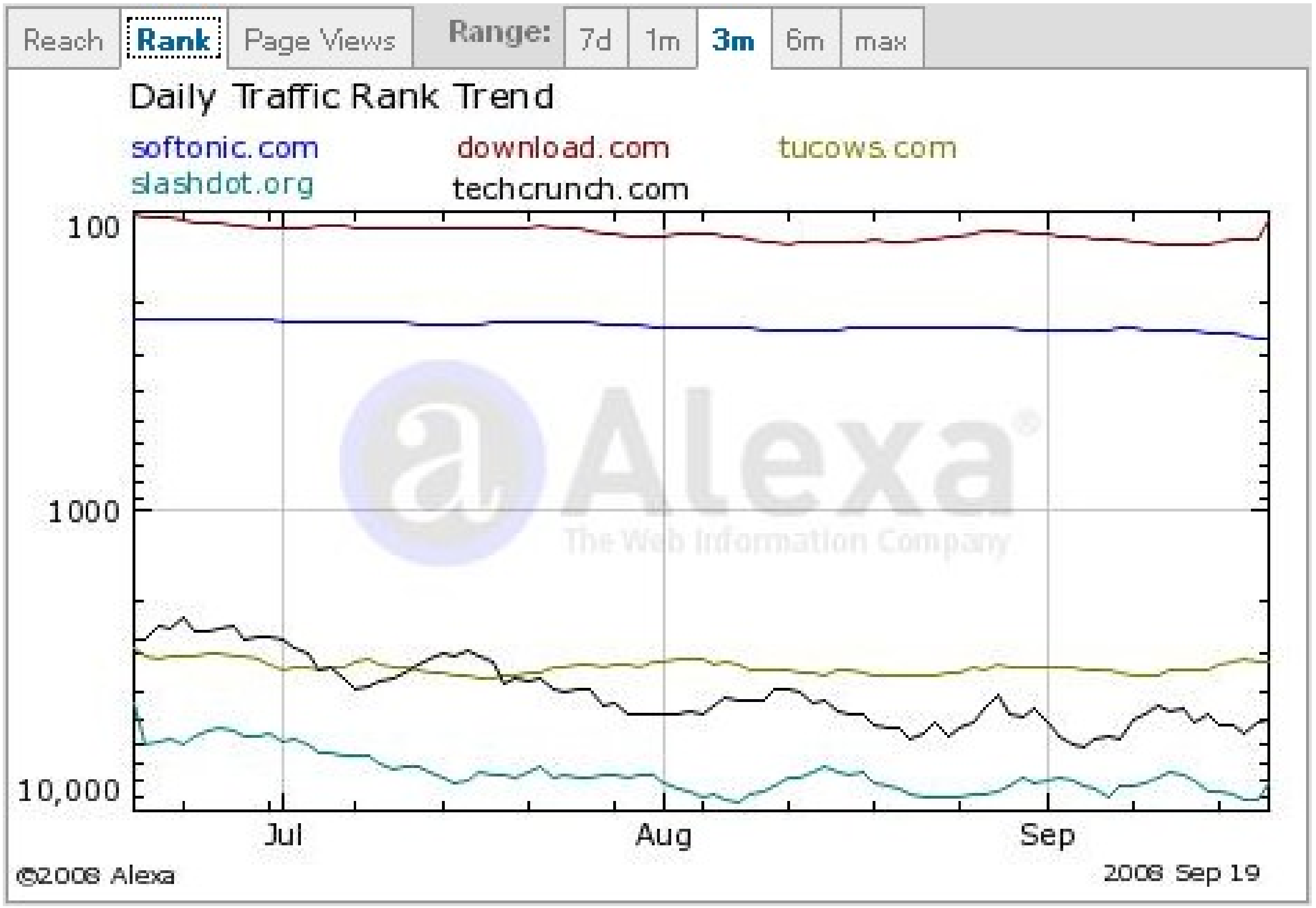
- Working with PHP since 2000
- From Germany, living in Spain, speaking English
- Always has some crazy projects in development
- Senior PHP developer and working for Softonic

- Contact: jens.bierkandt@softonic.com

Some details about Softonic

- Softonic is a software download portal founded in 1997
- It offers reviews, ratings, user comparison of programs, commenting functions and more
- It has its origins in Spain
- Now available in Spanish, English, German, French, Italian, Portuguese and more to come
- We have over 10 Million page views per day
- More than 1.3 Million downloads a day
- More than 100.000 sessions at the same time
- Over 100 servers, over 100 employees, ~1 SPE (server per employee)
- We love PHP

Some statistics about Softonic



What’s the problem

- Imagine you just launched a fresh application
- People actually love what you’ve created (finally!)
- They post it in their blogs
- It gets picked up in online magazines, newspapers, TV...
- And your server starts crying for help
- Just before being listed at Slashdot

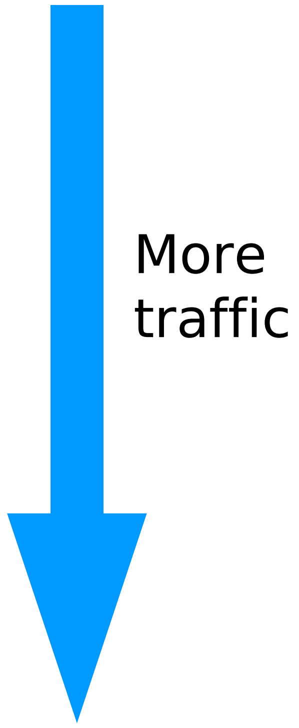
- This session will give you an overview about how you can keep your site running while getting more traffic. Where possible we try to use PHP to archive this goal

What you will learn today

- Error detection
  - Application wide
  - Server wide
- Using PHP with accelerators
- MySQL database replication
- Caching systems
- Webserver replication
- Session sharing
- High volume searches
- Problems/Solutions

How to handle more traffic

- Get webspace
- Get a webserver
- Install a PHP Accelerator
- Optimize SQL queries and PHP code
- Get an extra database server
- Cache DB results and HTML snippets
- Get server for static content (images)
- Get replication database
- Get more webservers
- Share sessions
- Use DB independent full text search
- Add servers where necessary



How to detect errors

“To fix an error, you need to know that you have an error.”

We distinguish between:

- Application wide error detection
- Server wide error detection

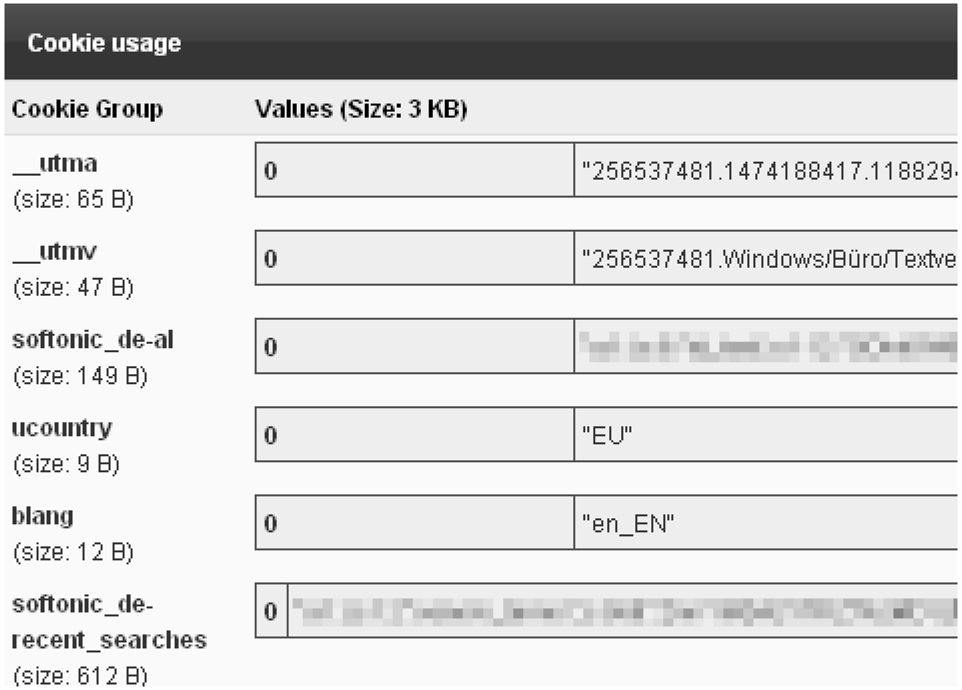
Application wide error detection

- Log all PHP errors/warnings in a logfile. Not all PHP errors and warnings show up while developing software. We can also trace `trigger_error()` events
- Log slow SQL queries in a logfile. Some queries are only slow when executed with special user content
- Log all SQL queries that fail. This gives you an idea if you sanitize correctly user input or pass correct values from your models

Create a debug console

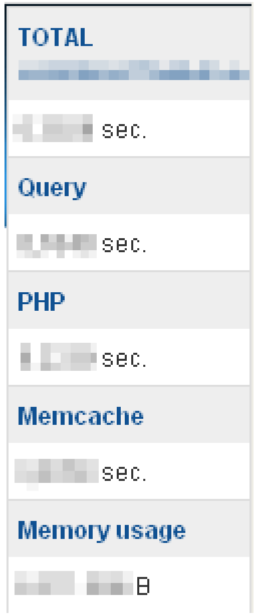
- Create debug output for every page containing:
  - Time needed by PHP to build the page
  - Detailed MySQL debug:
    - Time needed for each SQL query
    - The SQL query itself and the EXPLAIN of the SQL query
    - The result set
    - Trace where the query has been launched in the script
  - Caching info, session variables, everything you use
- Bottlenecks and bugs can be found easily

Examples from Softonic

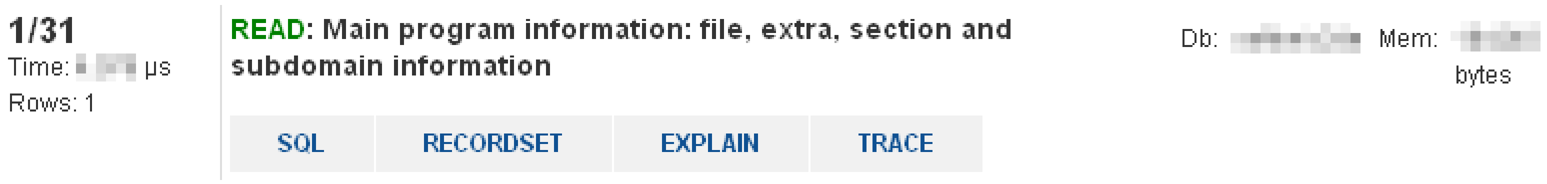


Cookie usage

Cookie Group	Values (Size: 3 KB)	
__utma (size: 65 B)	0	"256537481.1474188417.118829
__utmv (size: 47 B)	0	"256537481.Windows/Büro/Textve
softonic_de-al (size: 149 B)	0	[illegible]
ucountry (size: 9 B)	0	"EU"
blang (size: 12 B)	0	"en_EN"
softonic_de-recent_searches (size: 612 B)	0	[illegible]



TOTAL
[illegible] sec.
Query
[illegible] sec.
PHP
[illegible] sec.
Memcache
[illegible] sec.
Memory usage
[illegible] B



1/31
Time: [illegible] µs
Rows: 1

READ: Main program information: file, extra, section and subdomain information

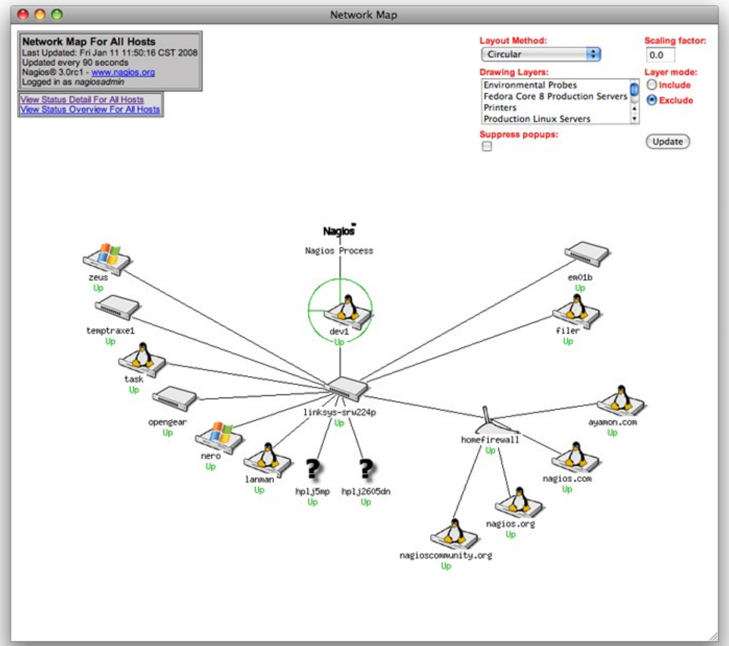
Db: [illegible] Mem: [illegible] bytes

Server wide error detection

- Server health checks with Nagios
  - Important to know if the servers work 24/7
  - Be informed immediately if a server crashes by SMS, E-Mail and alarm sounds
- Performance checks with Cacti
  - Watching for example the load average, outgoing traffic, queries per second etc.
  - See trends
  - Get an idea when to throw new servers to the application (e.g. more database slaves...)

Nagios

- Nagios is a host and service monitoring program.
- It can check all your services and alerts you if something is not running.

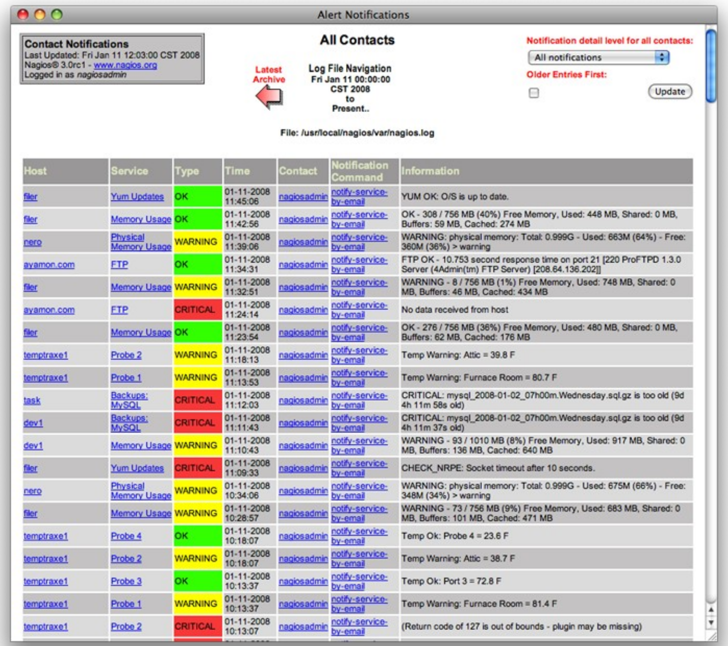


Network Map

Network Map For All Hosts
Last Updated: Fri Jan 11 11:50:16 CST 2008
Updated every 90 seconds
Nagios® 3.0rc1 - www.nagios.org
Logged in as *nagiosadmin*

View Status Detail For All Hosts
View Status Overview For All Hosts

Layout Method: Circular
Scaling factor: 0.0
Drawing Layers: Environmental Probes, Fedora Core 8 Production Servers, Printers, Production Linux Servers
Layer mode: Include [✓ Exclude]
Suppress popups: ☐



Alert Notifications

Contact Notifications
Last Updated: Fri Jan 11 12:03:00 CST 2008
Nagios® 3.0rc1 - www.nagios.org
Logged in as *nagiosadmin*

All Contacts

Latest Archive

Log File Navigation
Fri Jan 11 00:00:00
CST 2008
to
Present..

Notification detail level for all contacts: All notifications
Older Entries First: ☐

File: /usr/local/nagios/var/nagios.log

Host	Service	Type	Time	Contact	Notification Command	Information
filer	Yum Updates	OK	01-11-2008 11:45:06	nagiosadmin	notify-service-by-email	YUM OK: O/S is up to date.
filer	Memory Usage	OK	01-11-2008 11:42:56	nagiosadmin	notify-service-by-email	OK - 308 / 756 MB (40%) Free Memory, Used: 448 MB, Shared: 0 MB, Buffers: 59 MB, Cached: 274 MB
nero	Physical Memory Usage	WARNING	01-11-2008 11:39:06	nagiosadmin	notify-service-by-email	WARNING: physical memory: Total: 0.999G - Used: 663M (64%) - Free: 360M (36%) > warning
ayamon.com	FTP	OK	01-11-2008 11:34:31	nagiosadmin	notify-service-by-email	FTP OK - 10.753 second response time on port 21 [220 ProFTPD 1.3.0 Server (4Admin(tm) FTP Server) [208.64.136.202]]
filer	Memory Usage	WARNING	01-11-2008 11:32:51	nagiosadmin	notify-service-by-email	WARNING - 8 / 756 MB (1%) Free Memory, Used: 748 MB, Shared: 0 MB, Buffers: 46 MB, Cached: 434 MB
ayamon.com	FTP	CRITICAL	01-11-2008 11:24:14	nagiosadmin	notify-service-by-email	No data received from host
filer	Memory Usage	OK	01-11-2008 11:23:54	nagiosadmin	notify-service-by-email	OK - 276 / 756 MB (36%) Free Memory, Used: 480 MB, Shared: 0 MB, Buffers: 62 MB, Cached: 176 MB
temptraxe1	Probe 2	WARNING	01-11-2008 11:18:13	nagiosadmin	notify-service-by-email	Temp Warning: Attic = 39.8 F
temptraxe1	Probe 1	WARNING	01-11-2008 11:13:53	nagiosadmin	notify-service-by-email	Temp Warning: Furnace Room = 80.7 F
task	Backups: MySQL	CRITICAL	01-11-2008 11:12:03	nagiosadmin	notify-service-by-email	CRITICAL: mysql_2008-01-02_07h00m.Wednesday.sql.gz is too old (9d 4h 11m 58s old)
dev1	Backups: MySQL	CRITICAL	01-11-2008 11:11:43	nagiosadmin	notify-service-by-email	CRITICAL: mysql_2008-01-02_07h00m.Wednesday.sql.gz is too old (9d 4h 11m 37s old)
dev1	Memory Usage	WARNING	01-11-2008 11:10:43	nagiosadmin	notify-service-by-email	WARNING - 93 / 1010 MB (8%) Free Memory, Used: 917 MB, Shared: 0 MB, Buffers: 136 MB, Cached: 640 MB
filer	Yum Updates	CRITICAL	01-11-2008 11:09:33	nagiosadmin	notify-service-by-email	CHECK_NRPE: Socket timeout after 10 seconds.
nero	Physical Memory Usage	WARNING	01-11-2008 10:34:06	nagiosadmin	notify-service-by-email	WARNING: physical memory: Total: 0.999G - Used: 675M (66%) - Free: 348M (34%) > warning
filer	Memory Usage	WARNING	01-11-2008 10:28:57	nagiosadmin	notify-service-by-email	WARNING - 73 / 756 MB (9%) Free Memory, Used: 683 MB, Shared: 0 MB, Buffers: 101 MB, Cached: 471 MB
temptraxe1	Probe 4	OK	01-11-2008 10:18:07	nagiosadmin	notify-service-by-email	Temp Ok: Probe 4 = 23.6 F
temptraxe1	Probe 2	WARNING	01-11-2008 10:18:07	nagiosadmin	notify-service-by-email	Temp Warning: Attic = 38.7 F
temptraxe1	Probe 3	OK	01-11-2008 10:13:37	nagiosadmin	notify-service-by-email	Temp Ok: Port 3 = 72.8 F
temptraxe1	Probe 1	WARNING	01-11-2008 10:13:37	nagiosadmin	notify-service-by-email	Temp Warning: Furnace Room = 81.4 F
temptraxe1	Probe 2	CRITICAL	01-11-2008 10:13:07	nagiosadmin	notify-service-by-email	(Return code of 127 is out of bounds - plugin may be missing)

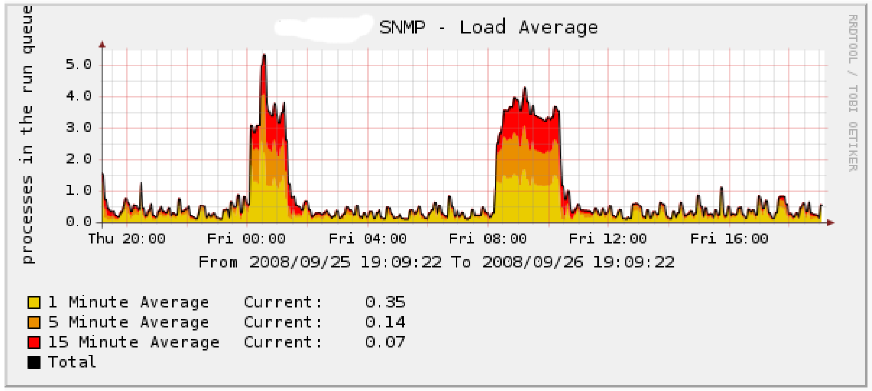
http://www.nagios.org/

Cacti

- Cacti enables you to see and analyse your server performance.
- Works with SNMP and is highly configurable with plugins
- Most important things to watch:
  - Load average
  - Outgoing traffic
  - Memory usage
  - MySQL queries per second
  - MySQL replication lag
  - Disk I/O

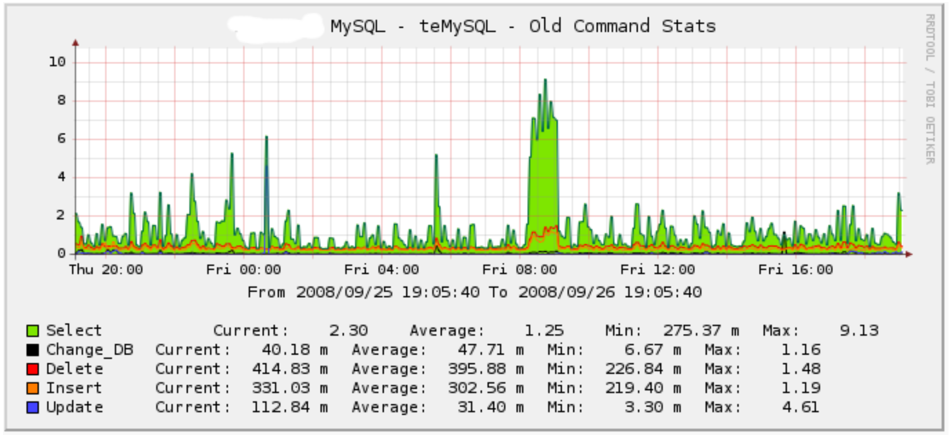
http://www.cacti.net/

Cacti screenshots



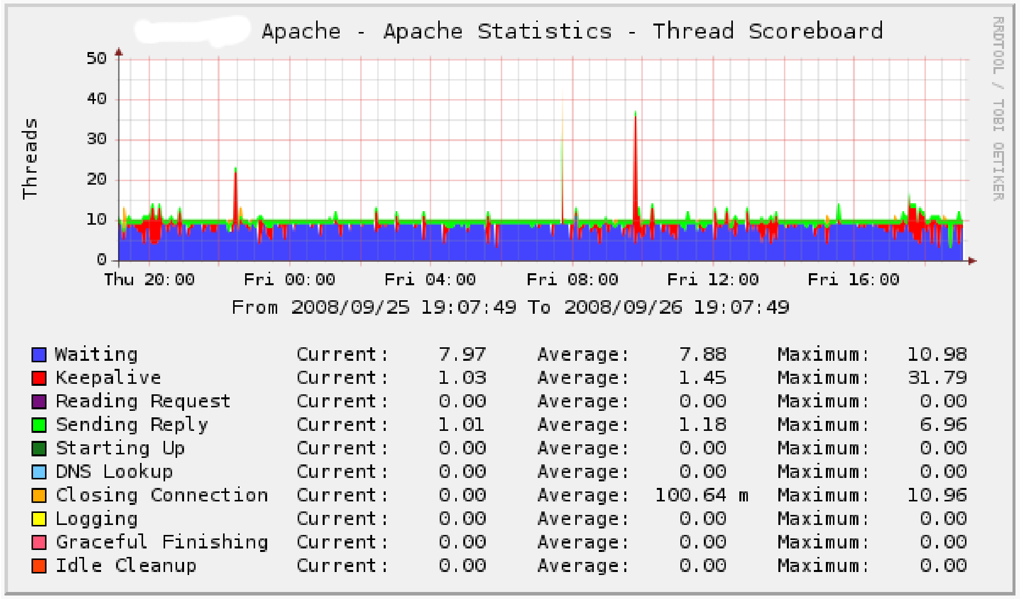
```
From 2008/09/25 19:09:22 To 2008/09/26 19:09:22

■ 1 Minute Average    Current:    0.35
■ 5 Minute Average    Current:    0.14
■ 15 Minute Average   Current:    0.07
■ Total
```



```
From 2008/09/25 19:05:40 To 2008/09/26 19:05:40

■ Select              Current:    2.30    Average:    1.25    Min:  275.37 m  Max:    9.13
■ Change_DB  Current:  40.18 m  Average:   47.71 m  Min:    6.67 m  Max:    1.16
■ Delete     Current: 414.83 m  Average:  395.88 m  Min:  226.84 m  Max:    1.48
■ Insert     Current: 331.03 m  Average:  302.56 m  Min:  219.40 m  Max:    1.19
■ Update     Current: 112.84 m  Average:   31.40 m  Min:    3.30 m  Max:    4.61
```

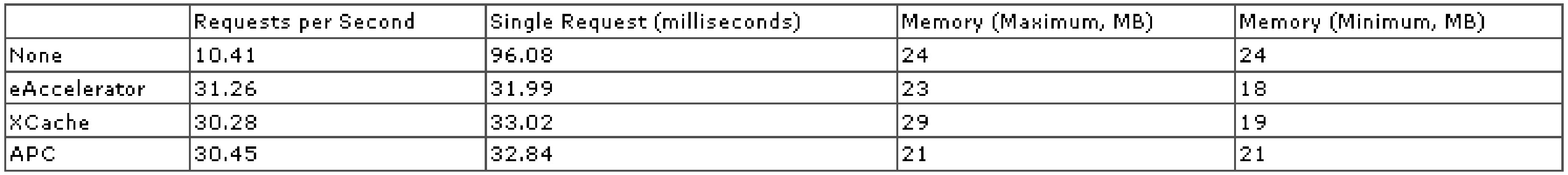


```
From 2008/09/25 19:07:49 To 2008/09/26 19:07:49

■ Waiting              Current:    7.97    Average:    7.88    Maximum:   10.98
■ Keepalive            Current:    1.03    Average:    1.45    Maximum:   31.79
■ Reading Request      Current:    0.00    Average:    0.00    Maximum:    0.00
■ Sending Reply        Current:    1.01    Average:    1.18    Maximum:    6.96
■ Starting Up          Current:    0.00    Average:    0.00    Maximum:    0.00
■ DNS Lookup           Current:    0.00    Average:    0.00    Maximum:    0.00
■ Closing Connection   Current:    0.00    Average:  100.64 m  Maximum:   10.96
■ Logging              Current:    0.00    Average:    0.00    Maximum:    0.00
■ Graceful Finishing   Current:    0.00    Average:    0.00    Maximum:    0.00
■ Idle Cleanup         Current:    0.00    Average:    0.00    Maximum:    0.00
```

PHP accelerators

- Op-code caches speed up PHP applications by parsing and tokenizing PHP scripts once, and executing them faster for every subsequent request
- For example APC, eaccelerator and XCache
- Easy to install. They normally work completely transparent for the developer
- Must have, use it as first aid for slow applications



	Requests per Second	Single Request (milliseconds)	Memory (Maximum, MB)	Memory (Minimum, MB)
None	10.41	96.08	24	24
eAccelerator	31.26	31.99	23	18
XCache	30.28	33.02	29	19
APC	30.45	32.84	21	21

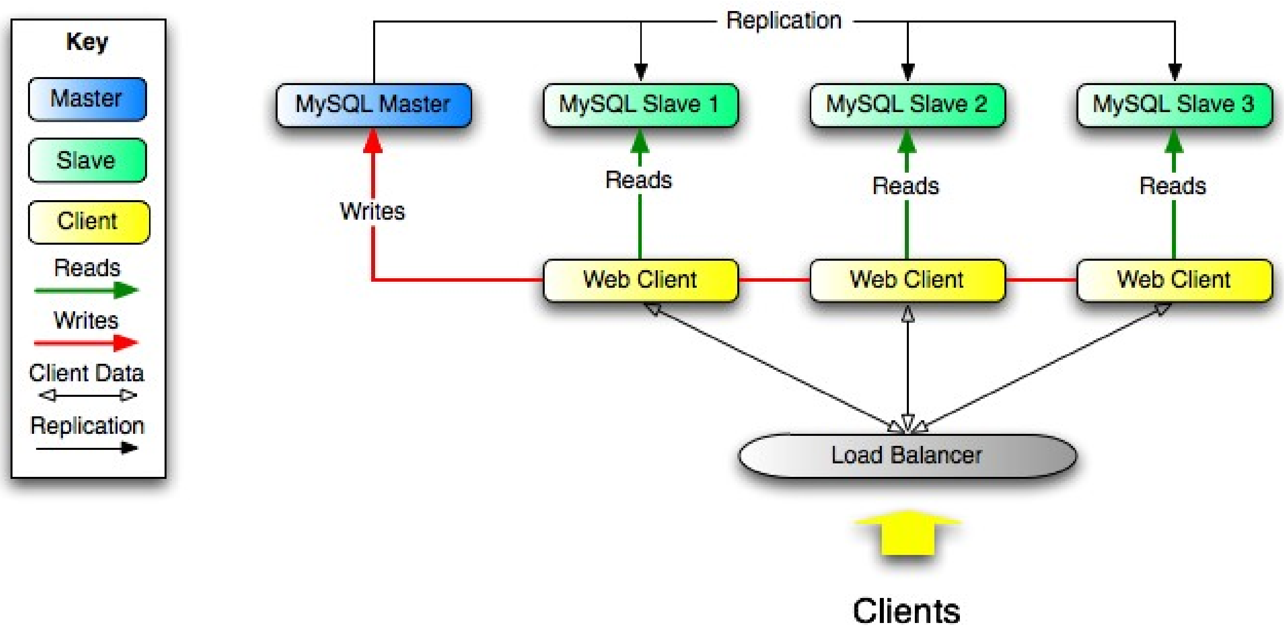
Source: http://2bits.com/articles/benchmarking-drupal-with-php-op-code-caches-apc-eaccelerator-and-xcache-compared.html

Alternative PHP Cache

- Caches compiled PHP scripts. Example: User A requests a script. It gets compiled and saved by APC. User B requests this file again and retrieves it directly from the cache
- APC is also a general purpose PHP shared memory store. You can put e.g. database results and HTML snippets to the APC cache and retrieve them later without bothering the database
- Saves data locally, in contrast to memcached
- Much faster than distributed caches over several servers

MySQL replication basics

- A typical website normally has lots of reads from the database and a few writes
- All writes go to the master, all reads to the the slaves



Source: MySQL website

How does a SELECT work

- User requests a page
- The PHP script decides which DB server to connect to for reading. This is based on the server weight and the server is checked for availability
- All SELECT queries are then executed on the same DB server until the script ends
- This functionality needs a well designed database class to make the calls transparent for the rest of the application

How does a WRITE work

- If a write query is needed, the database class automatically connects to the master database server
- The query is executed and the MySQL master automatically distributes the data to the replication slaves

Conclusion for MySQL replication

- MySQL replication is pretty easy to set up
- Big performance boost for web applications
- Problems:
    - More hardware is needed. More hardware means more hardware failures
    - A replication lag can occur. Replication slaves can’t catch the new data from the master instantly. They always need more or less time, depending on the load of the servers and the network. Therefore a SELECT from a replication slave is possibly getting old data before the slave gets updated information from the master
- MySQL provides also a SQL query cache which is disabled by default

Why using a server cache

- Generally it is not necessary to always regenerate a website completely when a user visits it
- A better approach is to execute the script once, do the selects from the DB once, create the HTML output and then save it on the server for the next user
- It’s possible to cache whole pages, HTML snippets or data (e.g. DB results)
- Implementation in the application can be complex

Using memcached

- Best practise is to install memcached on every webserver you have (Webserver is CPU intensive, memcached is memory intensive)
- A PHP extension for memcached exists and needs to be installed and configured
- When you launch e.g. a database query, always check in advance if there is a cached version available
- This drops the load of the DB servers significantly
- Be sure that the webserver has enough memory to always serve the cache requests from there

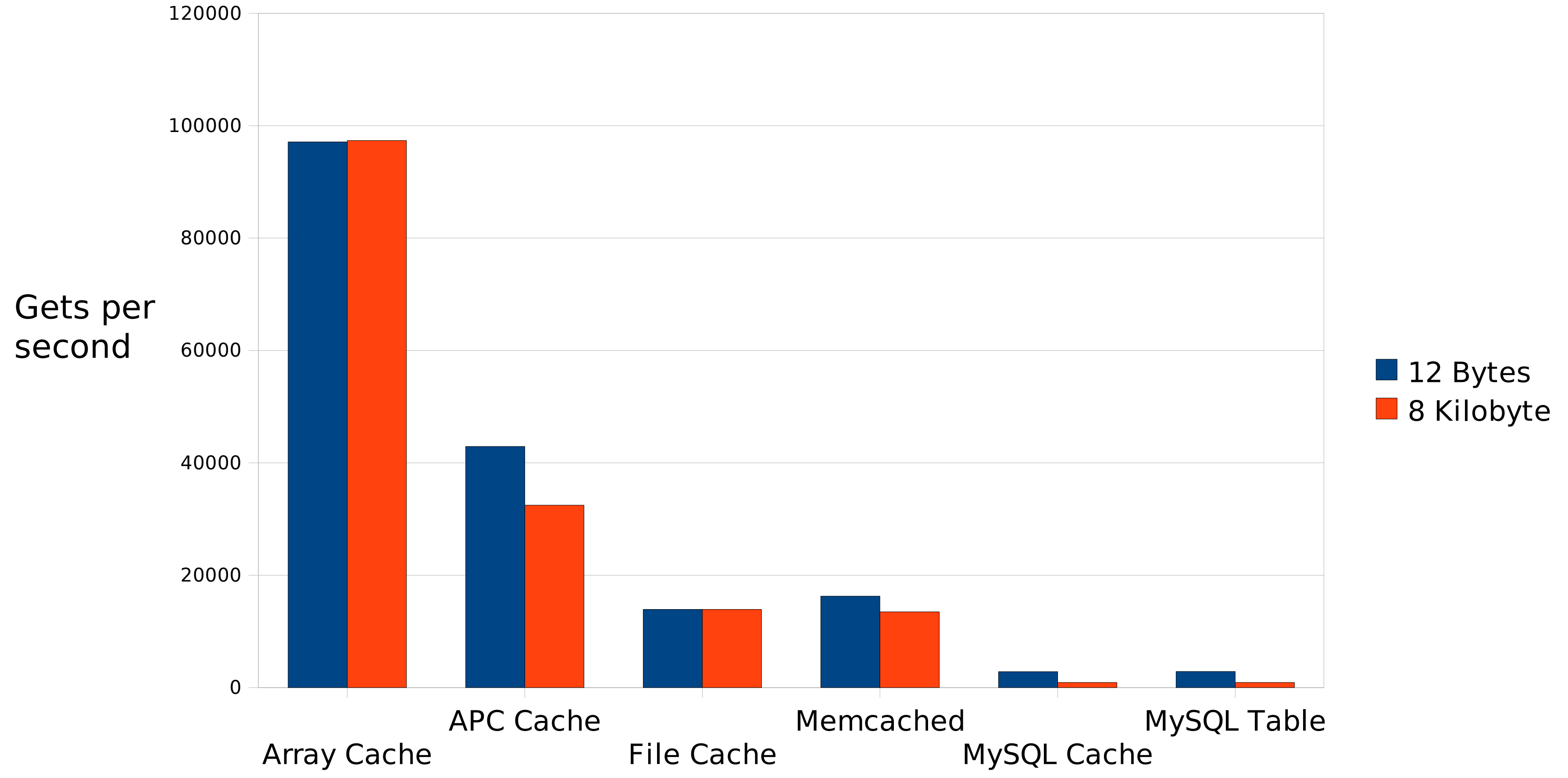
How does memcached work

- The memcached client knows about all available memcached servers
- When a user first requests a webpage from one of your webservers, the data is received from the database and send with a hash by the client to memcached
- If another user requests the same data, the memcached client retrieves the data from the correct memcached server
- If one memcached dies, the requests are remapped to the other available servers and stored/received from them

Caching problems

- If the content of the database changes, a manual deletion of the cache is necessary. If this is not implemented very well, the user might see old data
- If for any reason the memcache daemons need to be restarted or the cache needs to be completely deleted, you will see instantly a very high load on the database servers. This load can exceed the capacity of the servers and your application fails to start. In this case it is necessary to prefill the cache slowly before going online again

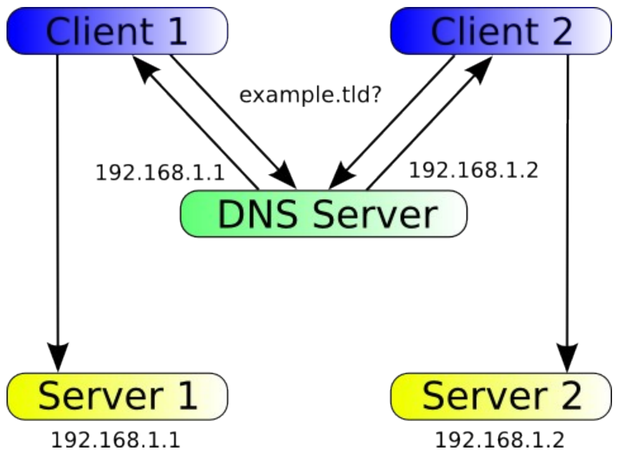
Caching benchmarks



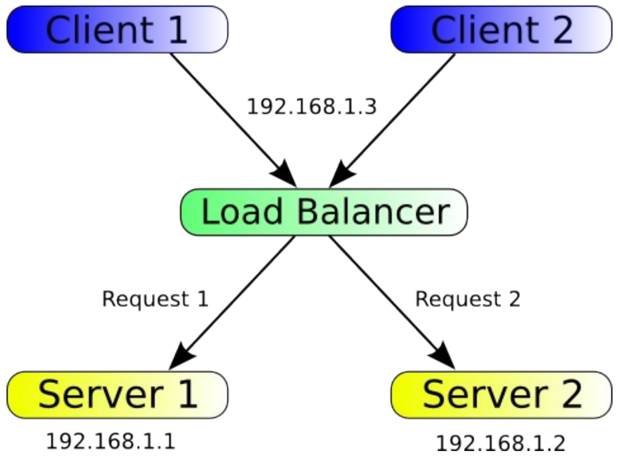
Benchmark tests with 10.000 cycles

Webserver replication basics

- Requests are distributed to different servers
- Several possibilities
  - Round Robin (DNS based)
  - Load Balancer (hardware/software based)



Round Robin



Load Balancer

Webserver replication issues

- The session data is normally saved locally. What happens, if a user gets data from his first request from server A and with the second request from server B?
  - A separate session handling must be used
- Distribution needed if a user uploads content that is normally not stored in the database (e.g. images)
  - Dedicated server for content uploaded by users
- Failover needed if a server crashes
  - Can be handled with a load balancer

Session basics

- A session is used when you store e.g. temporary data of a user on the server side. The session is identified by a unique string in a cookie or URL passed from the client to the server
- In PHP session data is by default stored in the file system
- Can be customized in PHP to store session data e.g. in a database, xml-file... with the function call `session_set_save_handler()`

Sharing sessions

- Using the local file system for sessions is not useful when having multiple webservers
- Using a shared file for saving sessions is not recommended because of performance issues
- Using a database for session storage puts even more load on the database servers
- Solution: Sharedance

Sharedance

- Sharedance is a distributed object cache
- A distributed cache spreads it's data across multiple machines
- Great for saving sessions webserver independent
- Pretty much like memcached

Sharedance versus memcached

- Sharedance can save session data also to the hard disk, memcached uses only memory
- Old session data could be thrown out if the cache gets full in memcached. In Sharedance it is kept until it gets deleted manually (session destroy) or by a given expiry time (GC)
- If a memcached server goes down, the cache data is lost on this server
- Both use libevent, both are daemons
- Sharedance has some more configuration parameters

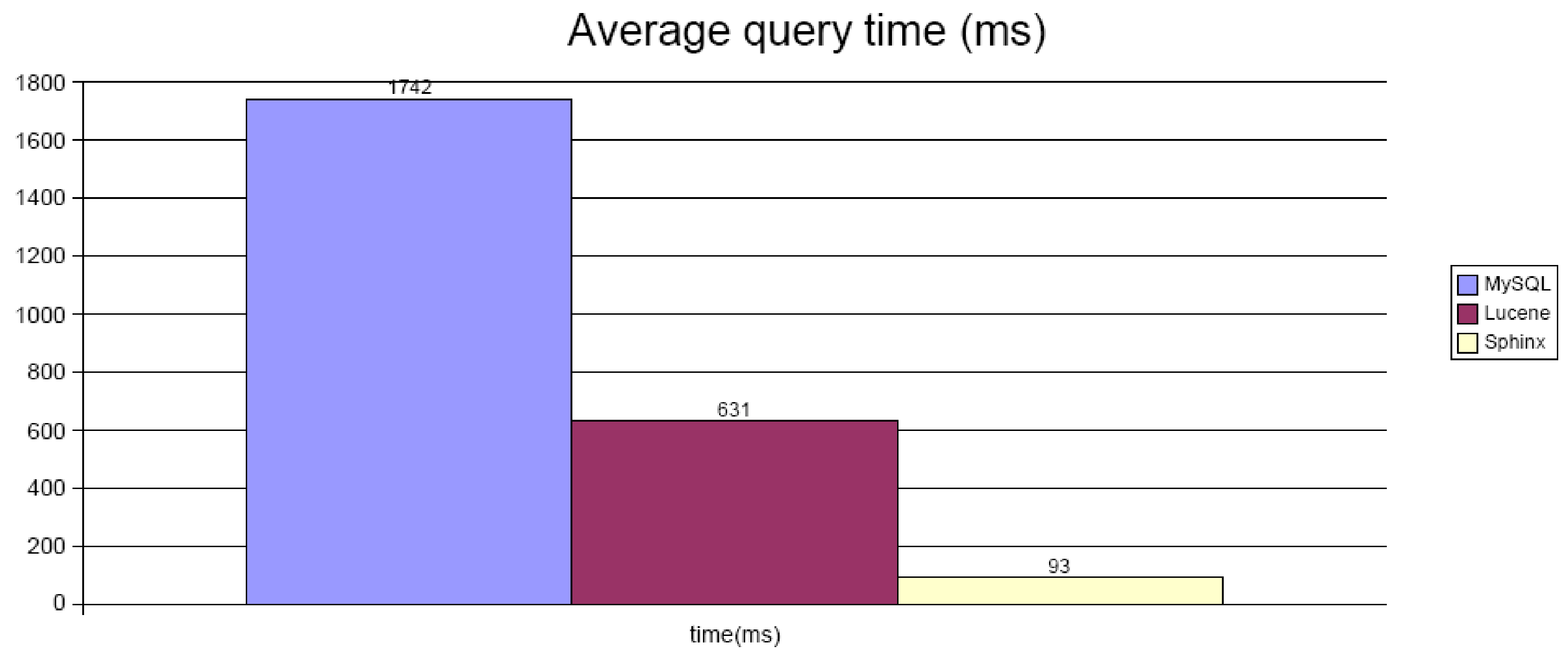
Searching basics

- Searching and finding content on a website is essential for internet users
- Most websites offer the possibility to browse through their data (e.g. products, reviews, text in general...)
- Depending on the amount of the users and data, this can be quite challenging for a database
- We need a full text search tool that's fast and does not put more load on the DB

Sphinx

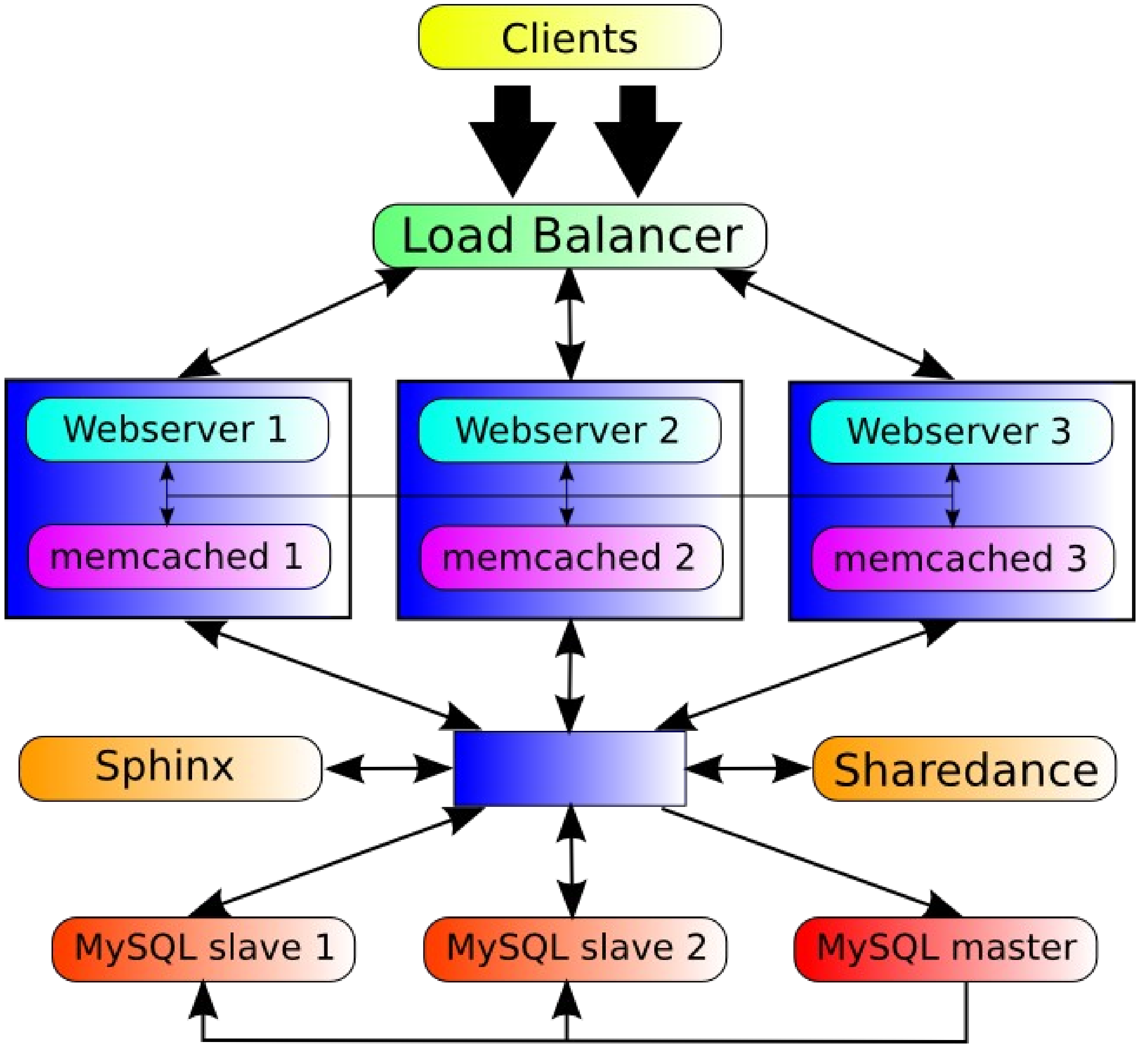
- Sphinx is a full-text search engine
- Sphinx was especially designed to integrate well with SQL databases and scripting languages
- High indexing speed (up to 10 MB/sec on modern CPUs)
- High search speed (average query is under 0.1 sec on 2-4 GB text collections)
- High scalability (up to 100 GB of text, up to 100 M documents on a single CPU)

Search engine benchmarks



Source: http://www.mysqlperformanceblog.com/files/presentations/EuroOSCON2006-High-Performance-FullText-Search.pdf

The big picture



What's left

- Create a filesystem inside the server memory (tmpfs)
- Create intermediate data with cronjobs
- Use extra servers for static content (e.g. images, CSS files...)
- Tweak my.cnf, httpd.conf and kernel parameters
- Add 'noatime' in /etc/fstab on your web and data drives to prevent disk writes on every read
- Benchmarking. All tools mentioned could be replaced by similar programs
- And much more...

Downside

- More servers mean also more hardware failure
- Costs quite some money to buy, configure and maintain
- Everything should be set up redundant, avoiding a single point of failure

That’s it!

- Questions?
- You can get these slide from here: http://www.bierkandt.org/php_barcelona2008.pdf
- The video will be online soon
- Thank you for joining and hope to see you again! ;-)

- Contact: jens.bierkandt@softonic.com