Tactics for using PHP in a high-load site

Asked 16 years, 4 months ago Modified 1 year, 8 months ago

Viewed 28k times Part of PHP Collective

253

Before you answer this I have never developed anything popular enough to attain high server loads. Treat me as (sigh) an alien that has just landed on the planet, albeit one that knows PHP and a few optimisation techniques.

I'm developing a tool in **PHP** that could attain quite a lot of users, if it works out right. However while I'm fully capable of developing the program I'm pretty much clueless when it comes to making something that can deal with huge traffic. So here's a few questions on it (feel free to turn this question into a resource thread as well).

Databases

At the moment I plan to use the MySQLi features in PHP5. However how should I setup the databases in relation to users and content? Do I actually *need* multiple databases? At the moment everything's jumbled into one database - although I've been considering spreading user data to one, actual content to another and finally core site content (template masters etc.) to another. My reasoning behind this is that sending queries to different databases

will ease up the load on them as one database = 3 load sources. Also would this still be effective if they were all on the same server?

Caching

I have a template system that is used to build the pages and swap out variables. Master templates are stored in the database and each time a template is called its cached copy (a html document) is called. At the moment I have two types of variable in these templates - a static var and a dynamic var. Static vars are usually things like page names, the name of the site - things that don't change often; dynamic vars are things that change on each page load.

My question on this:

Say I have comments on different articles. Which is a better solution: store the simple comment template and render comments (from a DB call) each time the page is loaded or store a cached copy of the comments page as a html page - each time a comment is added/edited/deleted the page is recached.

Finally

Does anyone have any tips/pointers for running a high load site on PHP. I'm pretty sure it's a workable language

to use - Facebook and Yahoo! give it great precedence - but are there any experiences I should watch out for?

PHP php performance high-load

Share

Improve this question

Follow

edited Apr 16, 2023 at 11:29

danronmoon

3,873 ● 5 ● 35 ● 58

asked Aug 23, 2008 at 22:37

Ross

47k ● 39 ● 123 ● 173

12 3.5 years later and I can't even remember what I was working on, I'd like to know what I thought was so cool too :) – Ross Dec 9, 2011 at 2:28

9 Let this be a lesson to you about premature optimization :) – Rimu Atkinson Dec 13, 2012 at 6:16

24 Answers

Sorted by: Highest score (default)

93

No two sites are alike. You really need to get a tool like jmeter and benchmark to see where your problem points will be. You can spend a lot of time guessing and improving, but you won't see real results until you measure and compare your changes.

For example, for many years, the MySQL query cache was the solution to all of our performance problems. If


your site was slow, MySQL experts suggested turning the query cache on. It turns out that if you have a high write load, the cache is actually crippling. If you turned it on without testing, you'd never know.

And don't forget that you are never done scaling. A site that handles 10req/s will need changes to support 1000req/s. And if you're lucking enough to need to support 10,000req/s, your architecture will probably look completely different as well.

Databases

- Don't use MySQLi -- PDO is the 'modern' OO database access layer. The most important feature to use is placeholders in your queries. It's smart enough to use server side prepares and other optimizations for you as well.
- You probably don't want to break your database up at this point. If you do find that one database isn't cutting, there are several techniques to scale up, depending on your app. Replicating to additional servers typically works well if you have more reads than writes. Sharding is a technique to split your data over many machines.

Caching

- You probably don't want to cache in your database. The database is typically your bottleneck, so adding more IO's to it is typically a bad thing. There are several PHP caches out there that accomplish similar things like APC and Zend.
- Measure your system with caching on and off. I bet your cache is heavier than serving the pages straight.
- If it takes a long time to build your comments and article data from the db, integrate memcache into your system. You can cache the query results and store them in a memcached instance. It's important to remember that retrieving the data from memcache must be faster than assembling it from the database to see any benefit.
- If your articles aren't dynamic, or you have simple dynamic changes after it's generated, consider writing out html or php to the disk. You could have an index.php page that looks on disk for the article, if it's there, it streams it to the client. If it isn't, it generates the article, writes it to the disk and sends it to the client. Deleting files from the disk would cause pages to be re-written. If a comment is added to an article, delete the cached copy -- it would be regenerated.

answered Aug 23, 2008 at 23:03

Gary Richardson

16.4k ● 10 ● 54 ● 48

11 @writing to disk. You could even ditch the index.php and let Apache do the work for you, so that index.php is only called, if the path doesn't exist. You'd use mode_rewrite for this. – troelskn Sep 29, 2008 at 10:47

5 -1, PDO is significantly slower than the MySQLi or even the MySQL extension. – Alix Axel Sep 10, 2009 at 0:15

4 PDO was much slower than mysqli and didn't work right for nested queries for me. Mysqli also supports server side prepares and bound parameters just like PDO. – Daren Schwenke Oct 14, 2009 at 3:23

5 I can't believe this was accepted as an answer. Its not very good. – symcbean Mar 26, 2010 at 16:05

1 about:caching - images, css, htm and js will help, turn off cookies on images too! – Talvi Watia Jun 30, 2010 at 23:53

69

I'm a lead developer on a site with over 15M users. We have had very little scaling problems because we planned for it EARLY and scaled thoughtfully. Here are some of the strategies I can suggest from my experience.

SCHEMA First off, denormalize your schemas. This means that rather than to have multiple relational tables, you should instead opt to have one big table. In general, joins are a waste of precious DB resources because doing multiple prepares and collation burns disk I/O's. Avoid them when you can.

The trade-off here is that you will be storing/pulling redundant data, but this is acceptable because data and

intra-cage bandwidth is very cheap (bigger disks) whereas multiple prepare I/O's are orders of magnitude more expensive (more servers).

INDEXING Make sure that your queries utilize at least one index. Beware though, that indexes will cost you if you write or update frequently. There are some experimental tricks to avoid this.

You can try adding additional columns that aren't indexed which run parallel to your columns that are indexed. Then you can have an offline process that writes the non-indexed columns over the indexed columns in batches. This way, you can control better when mySQL will need to recompute the index.

Avoid computed queries like a plague. If you must compute a query, try to do this once at write time.

CACHING I highly recommend Memcached. It has been proven by the biggest players on the PHP stack (Facebook) and is very flexible. There are two methods to doing this, one is caching in your DB layer, the other is caching in your business logic layer.

The DB layer option would require caching the result of queries retrieved from the DB. You can hash your SQL query using md5() and use that as a lookup key before going to database. The upside to this is that it is pretty easy to implement. The downside (depending on implementation) is that you lose flexibility because you're

treating all caching the same with regard to cache expiration.

In the shop I work in, we use business layer caching, which means each concrete class in our system controls its own caching schema and cache timeouts. This has worked pretty well for us, but be aware that items retrieved from DB may not be the same as items from cache, so you will have to update cache and DB together.

DATA SHARDING Replication only gets you so far. Sooner than you expect, your writes will become a bottleneck. To compensate, make sure to support data sharding early as possible. You will likely want to shoot yourself later if you don't.

It is pretty simple to implement. Basically, you want to separate the key authority from the data storage. Use a global DB to store a mapping between primary keys and cluster ids. You query this mapping to get a cluster, and then query the cluster to get the data. You can cache the hell out of this lookup operation which will make it a negligible operation.

The downside to this is that it may be difficult to piece together data from multiple shards. But, you can engineer your way around that as well.

OFFLINE PROCESSING Don't make the user wait for your backend if they don't have to. Build a job queue and move any processing that you can offline, doing it separate from the user's request.

Share Improve this answer

Follow

answered Jan 20, 2009 at 1:12

thesmart

3,063 ● 2 ● 33 ● 34

10 +1 Hands down, this should be the accepted answer. It's interesting that everything I've ever read about building databases always says "normalize all the data as much as possible" without mentioning the performance hit of doing joins. I've always intuitively felt that joins (especially multiple) added a lot of overhead but haven't heard any say it explicitly until now. I wish I better understood what you were talking about controlling when MySQL computes the indexes, it sounds like a very interesting hack. – Evan Plaice Feb 14, 2011 at 12:10

Data sharding is essential for databases that grow too big. Google (the company not the search engine) has a lot of interesting things to say about implementing sharding schemes. Offline processing is also huge when it comes down to limiting the number of database writes (and limiting the number of table index recalculations). I've seen lots of blogs (and I think even Stack Overflow) use this technique for their user-generated comment/feedback systems. – Evan Plaice Feb 14, 2011 at 12:16

1 Thank you for the comments. It is amazing that some argue for profiling middle-tier code when the VAST amount of execution time is spent in either data I/O or client-server I/O. An ubber complicated optimization saving 20% off execution-time of a PHP process that takes 40ms is pointless compared to simple 5% savings off of a 1s database query. – thesmart Feb 16, 2011 at 23:01

I've worked on a few sites that get millions/hits/month backed by PHP & MySQL. Here are some basics:

42

1. Cache, cache, cache. Caching is one of the simplest and most effective ways to reduce load on your webserver and database. Cache page content, queries, expensive computation, anything that is I/O bound. Memcache is dead simple and effective.
2. Use multiple servers once you are maxed out. You can have multiple web servers and multiple database servers (with replication).
3. Reduce overall # of request to your webservers. This entails caching JS, CSS and images using expires headers. You can also move your static content to a CDN, which will speed up your user's experience.
4. Measure & benchmark. Run Nagios on your production machines and load test on your dev/qa server. You need to know when your server will catch on fire so you can prevent it.

I'd recommend reading Building Scalable Websites, it was written by one of the Flickr engineers and is a great reference.

Check out my blog post about scalability too, it has a lot of links to presentations about scaling with multiple languages and platforms: http://www.ryandoherty.net/2008/07/13/unicorns-and-scalability/

Share Improve this answer

Follow

answered Aug 23, 2008 at 22:54

Ryan Doherty

38.7k ● 4 ● 57 ● 62

1 +1 There's a lot of good info here. I've been researching more on this topic lately and your answer falls in line with everything I've read. Memcache, caching, CDN for static content, reducing requests; all good stuff. I would also add, generate hashes on static content files (if your behind a CDN/cache) server-side so the updated files have a unique signature in the cache. Also, combine static source files (css, javascript) on the fly (and cache them with filename hashes) to cut down on requests. Also, generate thumbs dynamically (and store them in the cache) – Evan Plaice Feb 14, 2011 at 11:58

Google has created an apache module called mod_pagespeed that can handle all of the file concatenations, minification, file renaming to include hash, etc for all static content. It'll should only add a little processing overhead to the servers initially until the caches (and CDN(s)) are populated with most of the content. Also, for security, it's generally a bad idea to put tables that are publicly accessible (users) in the same database as tables than handle the back-end (if for some reason one of the tables were to be hacked). – Evan Plaice Feb 14, 2011 at 11:58

40

Re: PDO / MySQLi / MySQLND

@gary

You cannot just say "don't use MySQLi" as they have different goals. PDO is almost like an abstraction layer (although it is not actually) and is designed to make it easy to use multiple database products whereas MySQLi is specific to MySQL conections. It is wrong to say that PDO is the modern access layer in the context of

comparing it to MySQLi because your statement implies that the progression has been mysql -> mysqli -> PDO which is not the case.

The choice between MySQLi and PDO is simple - if you need to support multiple database products then you use PDO. If you're just using MySQL then you can choose between PDO and MySQLi.

So why would you choose MySQLi over PDO? See below...

@ross

You are correct about MySQLnd which is the newest MySQL core language level library, however it is not a replacement for MySQLi. MySQLi (as with PDO) remains the way you would interact with MySQL through your PHP code. Both of these use libmysql as the C client behind the PHP code. The problem is that libmysql is outside of the core PHP engine and that is where mysqlnd comes in i.e. it is a Native Driver which makes use of the core PHP internals to maximise efficiency, specifically where memory usage is concerned.

MySQLnd is being developed by MySQL themselves and has recently landed onto the PHP 5.3 branch which is in RC testing, ready for a release later this year. You will then be able to use MySQLnd with MySQLi...but not with PDO. This will give MySQLi a performance boost in many areas (not all) and will make it the best choice for MySQL

interaction if you do not need the abstraction like capabilities of PDO.

That said, MySQLnd is now available in PHP 5.3 for PDO and so you can get the advantages of the performance enhancements from ND into PDO, however, PDO is still a generic database layer and so will be unlikely to be able to benefit as much from the enhancements in ND as MySQLi can.

Some useful benchmarks can be found here although they are from 2006. You also need to be aware of things like this option.

There are a lot of considerations that need to be taken into account when deciding between MySQLi and PDO. It reality it is not going to matter until you get to rediculously high request numbers and in that case, it makes more sense to be using an extension that has been specifically designed for MySQL rather than one which abstracts things and happens to provide a MySQL driver.

It is not a simple matter of which is best because each has advantages and disadvantages. You need to read the links I've provided and come up with your own decision, then test it and find out. I have used PDO in past projects and it is a good extension but my choice for pure performance would be MySQLi with the new MySQLND option compiled (when PHP 5.3 is released).

edited May 23, 2017 at 12:17

Share Improve this answer

Follow

Community Bot

1 ● 1

answered Aug 24, 2008 at 14:17

davidmytton

39.2k ● 39 ● 89 ● 94

6 I switched from PDO to mysqli and regular queries started to execute exactly 2 times faster. – serg Nov 16, 2008 at 19:50

5 @serg: care to post some tests to confirm this ?, because I seriously doubt that simply switching from PDO to mysqli would give you such a speed boost. – Stann Mar 12, 2011 at 19:28

23

General

- Do not try to optimize before you start to see real world load. You might guess right, but if you don't, you've wasted your time.
- Use jmeter, xdebug or another tool to benchmark the site.
- If load starts to be an issue, either object or data caching will likely be involved, so generally read up on caching options (memcached, MySQL caching options)

Code

- Profile your code so that you know where the bottleneck is, and whether it's in code or the database

Databases

- Use MYSQLi if portability to other databases is not vital, PDO otherwise
- If benchmarks reveal the database is the issue, check the queries before you start caching. Use EXPLAIN to see where your queries are slowing down.
- After the queries are optimized and the database is cached in some way, you may want to use multiple databases. Either replicating to multiple servers or sharding (splitting the data over multiple databases/servers) may be appropriate, depending on the data, the queries, and the kind of read/write behavior.

Caching

- Plenty of writing has been done on caching code, objects, and data. Look up articles on APC, Zend Optimizer, memcached, QuickCache, JPCache. Do some of this before you really need to, and you'll be less concerned about starting off unoptimized.
- APC and Zend Optimizer are opcode caches, they speed up PHP code by avoiding reparsing and recompilation of code. Generally simple to install, worth doing early.
- Memcached is a generic cache, that you can use to cache queries, PHP functions or objects, or entire pages. Code must be specifically written to use it,

which can be an involved process if there are no central points to handle creation, update and deletion of cached objects.

- QuickCache and JPCache are file caches, otherwise similar to Memcached. The basic concept is simple, but also requires code and is easier with central points of creation, update and deletion.

Miscellaneous

- Consider alternative web servers for high load. Servers like lighthttp and nginx can handle large amounts of traffic in much less memory than Apache, if you can sacrifice Apache's power and flexibility (or if you just don't need those things, which often, you don't).
- Remember that hardware is surprisingly cheap these days, so be sure to cost out the effort to optimize a large block of code versus "let's buy a monster server."
- Consider adding the "MySQL" and "scaling" tags to this question

Share Improve this answer Follow

edited Sep 29, 2008 at 10:32

answered Sep 29, 2008 at 10:24

Paul Kroll

570 ● 2 ● 5

9

APC is an absolute must. Not only does it make for a great caching system, but the gain from the auto-cached PHP files is a godsend. As for the multiple database idea, I don't think you would get much out of having different databases on the same server. It may give you a bit of a gain in speed during query time, but I doubt the effort it would take to deploy and maintain the code for all three while making sure they are in sync would be worth it.

I also highly recommend running Xdebug to find bottlenecks in your program. It made optimization a breeze for me.

Share Improve this answer

Follow

answered Aug 23, 2008 at 22:45

tslocum

3,422 ● 5 ● 32 ● 33

9

Firstly, as I think Knuth said, "Premature optimization is the root of all evil". If you don't have to deal with these issues right now then don't, focus on delivering something that works correctly first. That being said, if the optimizations can't wait.

Try profiling your database queries, figure out what's slow and what happens alot and come up with an optimization strategy from that.

I would investigate Memcached as it's what a lot of the higher load sites use for efficiently caching content of all types, and the PHP object interface to it is quite nice.

Splitting up databases among servers and using some sort of load balancing technique (e.g. generate a random number between 1 and # redundant databases with necessary data - and use that number to determine which database server to connect to) can also be an excellent way to increase efficiency.

These have all worked out pretty well in the past for some fairly high load sites. Hope this helps to get you started :-)

Share Improve this answer

Follow

answered Aug 23, 2008 at 22:50

Eric Scrivner

1,849 ● 1 ● 19 ● 23

1 RequiredFullQuote: "We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil" – Alister Bulman Sep 12, 2011 at 9:05

RequiredReallyFullQuote: "Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%." – cHao May 21, 2013 at 10:18

For what it's worth, caching is DIRT SIMPLE in PHP even without an extension/helper package like memcached.

7

All you need to do is create an output buffer using `ob_start()`.

Create a global cache function. Call `ob_start`, pass the function as a callback. In the function, look for a cached version of the page. If exists, serve it and end.

If it doesn't exist, the script will continue processing. When it reaches the matching ob_end() it will call the function you specified. At that time, you just get the contents of the output buffer, drop them in a file, save the file, and end.

Add in some expiration/garbage collection.

And many people don't realize you can nest `ob_start()`/`ob_end()` calls. So if you're already using an output buffer to, say, parse in advertisements or do syntax highlighting or whatever, you can just nest another `ob_start/ob_end` call.

Share Improve this answer

Follow

edited Jan 15, 2016 at 15:32

A J

4,016 ● 14 ● 39 ● 54

answered Aug 27, 2008 at 20:32

Shane

+1 because it looks like an interesting idea. I don't know how well it works performance-wise – Sylver Apr 22, 2009 at 12:06

+1 because this is an interesting idea. Those callbacks could call my caching class for me! – Xeoncross Dec 6, 2009 at 20:24

6

Profiling your app with something like Xdebug (like tj9991 recommended) is definitely going to be a must. It doesn't make a whole lot of sense to just go around optimizing things blindly. Xdebug will help you find the real bottlenecks in your code so you can spend your optimization time wisely and fix chunks of code that are actually causing slow downs.

If you're using Apache, another utility that can help in testing is Siege. It will help you anticipate how your server and application will react to high loads by really putting it through its paces.

Any kind of opcode cache for PHP (like APC or one of the many others) will help a lot as well.

Share Improve this answer Follow

answered Aug 23, 2008 at 22:54

Bob Somers

7,306 ● 5 ● 43 ● 46

6

I run a website with 7-8 million page views a month. Not terribly much, but enough that our server felt the load. The solution we chose was simple: Memcache at the database level. This solution works well if the database load is your main problem.

We started out using Memcache to cache entire objects and the database results that were most frequently used. It did work, but it also introduced bugs (we might have avoided some of those if we had been more careful).

So we changed our approach. We built a database wrapper (with the exact same methods as our old database, so it was easy to switch), and then we subclassed it to provide memcached database access methods.

Now all you have to do is decide whether a query can use cached (and possibly out of date) results or not. Most of the queries run by the users are now fetched directly from Memcache. The exceptions are updates and inserts, which for the main website only happens because of logging. This rather simple measure reduced our server load by about 80%.

Share Improve this answer

Follow

answered Aug 26, 2008 at 9:38

Vegard Larsen

13k ● 14 ● 60 ● 102

5

> Thanks for the advice on PHP's caching extensions - could you explain reasons for using one over another? I've heard great things about memcached through IRC but have never heard of APC - what are your opinions on them? I assume using multiple caching systems is pretty counter-effective.

Actually, many do use APC and memcached together...

Share Improve this answer

Follow

answered Aug 24, 2008 at 14:26

ceejayoz
180k ● 41 ● 308 ● 380

4

It looks like I was wrong. MySQLi is still being developed. But according to the article, PDO_MySQL is now being contributed to by the MySQL team. From the article:

> The MySQL Improved Extension - mysqli - is the flagship. It supports all features of the MySQL Server including Charsets, Prepared Statements and Stored Procedures. The driver offers a hybrid API: you can use a procedural or object-oriented programming style based on your preference. mysqli comes with PHP 5 and up. Note that the End of life for PHP 4 is 2008-08-08.
>
> The PHP Data Objects (PDO) are a database access abstraction layer. PDO allows you to use the same API calls for various databases. PDO does not offer any degree of SQL abstraction. PDO_MYSQL is a MySQL driver for PDO. PDO_MYSQL comes with PHP 5. As of PHP 5.3 MySQL developers actively contribute to it. The PDO benefit of a unified API comes at the price that MySQL specific features, for example

> multiple statements, are not fully supported through the unified API.
>
> Please stop using the first MySQL driver for PHP ever published: ext/mysql. Since the introduction of the MySQL Improved Extension - mysqli - in 2004 with PHP 5 there is no reason to still use the oldest driver around. ext/mysql does not support Charsets, Prepared Statements and Stored Procedures. It is limited to the feature set of MySQL 4.0. Note that the Extended Support for MySQL 4.0 ends at 2008-12-31. Don't limit yourself to the feature set of such old software! Upgrade to mysqli, see also Converting_to_MySQLi. mysql is in maintenance only mode from our point of view.

To me, it seems the article is biased towards MySQLi. I suppose I'm biased towards PDO. I really like PDO over MySQLi. It's straight forward to me. The API is a lot closer to other languages I've programmed in. OO Database interfaces seem to work better.

I haven't come across any specific MySQL features that weren't available through PDO. I would be surprised if I ever did.

answered Aug 24, 2008 at 14:14

Gary Richardson

16.4k ● 10 ● 54 ● 48

3

PDO is also very slow and its API is pretty complicated. No one in their sane mind should use it if portability is not a concern. And let's face it, in 99% of all webapps it is not. You just stick with MySQL or PostrgreSQL, or whatever it is you are working with.

As for the PHP question and what to take into account. I think premature optimization is the root of all evil. ;) Get your application done first, try to keep it clean when it comes to programming, do a little documentation and write unit tests. With all of the above you will have no issues refactoring code when the time comes. But first you want to be done and push it out to see how people react to it.

Share Improve this answer

Follow

answered Aug 25, 2008 at 16:32

Till

22.4k ● 4 ● 60 ● 89

2

Sure pdo is nice, but there has been some controversy about it's performance versus mysql and mysqli, although it seems fixed now.

You should use pdo if you envision portability, but if not, mysqli should be the way. It has an OO interface, prepared statements, and most of what pdo offers (except, well, portability).

Plus, if performance is really needed, prepare for the (native mysql) MysqLnd driver in PHP 5.3, who will be

much more tightly integrated with php, with better performance and improved memory usage (and statistics for performance tuning).

Memcache is nice if you have clustered servers (and YouTube-like load), but i'd try out APC first too.

Share Improve this answer

Follow

answered Aug 24, 2008 at 13:55

Berzemus

3,658 ● 24 ● 32

2

A lot of good answers were given already, but I would like to point you to an alternate opcode cache called XCache. It is created by a lighty contributor.

Also, if you may need load balancing your database server in future, MySQL Proxy could very well help you to achieve this.

Both of those tools should plug into an existing application quite easily, so this optimization can be done when you need it, without too much hassle.

Share Improve this answer

Follow

answered Nov 16, 2008 at 19:07

hangy

10.9k ● 6 ● 45 ● 63

2

First question is how big do you really expect it to be? And how much do you plan on investing in your infrastructure. Since you feel the need to ask the question

here, I'm guessing that you expect to start small on a limited budget.

Performance is irrelevant if the site is not available. And for availability you need horizontal scaling. The minimum you can sensibly get away with is 2 servers, both running apache, php and mysql. Set up one DBMS as a slave to the other. Do all the writes on the master, and all the reads on the local database (whatever that is) - unless for some reason you need to read back the data you've just read (use master). Make sure you've got the machinery in place to automatically promote the slave and fence the master. Use round-robin DNS for the webserver addresses to give more affinity for the slave node.

Partitioning your data across different database nodes at this stage is a very bad idea - however you might want to consider splitting it across different databases on the same server (which will facilitate partitioning across nodes when you overtake facebook).

Do make sure you've got the monitoring and data analysis tools in place to measure your sites performance and identify bottlenecks. Most performance problems can be fixed by writing better SQL / fixing the database schema.

Keeping your template cache on the database is a dumb idea - the database should be a central common repository for structured data. Keep your template cache on the local filesystem of your webservers - it will be

available faster and won't slow down your database access.

Do use a op-code cache.

Spend plenty of time studying your site and its logs to understand why its going so slow.

Push as much caching as possible onto the client.

Use mod_gzip to compress everything you can.

C.

Share Improve this answer

Follow

answered Mar 26, 2010 at 16:19

symcbean
48.3k ● 6 ● 62 ● 97

2

My first piece of advice is to think about this issue and keep it in mind when designing the site but **don't go overboard**. It's often difficult to predict the success of a new site and I your time will be better spent getting up finished early and optimising it later.

In general, **Simple is fast**. Templates slow you down. Databases slow you down. Complex libraries slow you down. Layering templates over each other retrieving them from databases and parsing it in a complex library --> the time delays multiply with each other.

Once you have the basic site up and running **do tests** to show you where to spend your efforts. It's difficult to see

where to target. Often to speed things up you will have to unravel the complexity of the code, this makes it larger and harder to maintain, so you only want to do it where necessary.

In my experience establishing the database connection was relatively expensive. If you can get away with it, don't connect to the database for general visitors on the most trafficed pages like the front page to the site. Creating multiple database connections is madness with very little benefit.

Share Improve this answer

Follow

answered Jun 29, 2010 at 1:31

lod

31 ● 2

1

@Gary

> Don't use MySQLi -- PDO is the 'modern' OO database access layer. The most important feature to use is placeholders in your queries. It's smart enough to use server side prepares and other optimizations for you as well.

I'm loking over PDO at the moment and it looks like you're right - however I know that MySQL are developing the MySQLd extension for PHP - I think to succeed either MySQL or MySQLi - what do you think about that?

@Ryan, Eric, tj9991

Thanks for the advice on PHP's caching extensions - could you explain reasons for using one over another? I've heard great things about memcached through IRC but have never heard of APC - what are your opinions on them? I assume using multiple caching systems is pretty counter-effective.

I will definitely be sorting out some profiling testers - thank you very much for your recommendations on those.

Share Improve this answer

Follow

edited May 23, 2017 at 12:26

Community Bot

1 ● 1

answered Aug 24, 2008 at 12:38

Ross

47k ● 39 ● 123 ● 173

1

I don't see myself switching from MySQL anytime soon - so I guess I don't need the abstraction capabilities of PDO. Thanks for those articles DavidM, they've helped me a lot.

Share Improve this answer

Follow

answered Aug 24, 2008 at 14:25

Ross

47k ● 39 ● 123 ● 173

Look into mod_cache, an output cache for the Apache web server, simillar to the output caching in ASP.NET.

Yes, I can see that it's still experimental but it will be final someday.

1

Share Improve this answer

Follow

answered Aug 31, 2008 at 1:50

Andrei Rînea

20.7k ● 18 ● 121 ● 169

I can't believe no-one has already mentioned this: Modularisation and Abstraction. If you think your site is going to have to grow to lots of machines, you *must* design it so it can! That means stupid things like don't assume the database is on localhost. It also means things that are going to be a bother at first, like writing a database abstraction layer (like PDO, but much much lighter because it only does what you need it to do).

1

And it means things like working with a framework. You will need layers to your code so that you can later gain performance by refactoring the data-abstraction layer, for example, by teaching it that some objects are in a different database -- *and the code doesn't have to know or care*.

Finally, be careful of memory-intensive operations, for example, unnecessary string copying. If you can keep PHP's memory usage down, then you will get more

performance out of your webserver and this is something that will scale when you go to a load-balanced solution.

Share Improve this answer

Follow

answered Oct 29, 2008 at 23:43

staticsan

30.5k ● 5 ● 63 ● 73

1

If you are working with large amounts of data, and caching isn't cutting it, look into Sphinx. We've had great results with using SphinxSearch not only for better text searching, but also as a data retrieval replacement for MySQL when dealing larger tables. If you use SphinxSE (MySQL plugin), it surpassed our performance gains we had from caching several times over, and application-implementation is a sinch.

Share Improve this answer

Follow

answered Apr 15, 2009 at 16:49

SirAdrian

1

The points made about cache are spot-on; it is the least complicated and most important part of building an efficient application. I'd like to add that while memcached is great, APC is about five times faster if your application lives on a single server.

The "Cache Performance Comparison" post at the MySQL performance blog has some interesting benchmarks on the subject -

http://www.mysqlperformanceblog.com/2006/08/09/cache-performance-comparison/.

Share Improve this answer

Follow

answered Feb 2, 2010 at 0:11

Johannes Gorset

8,785 ● 4 ● 38 ● 34

0

1. Test each query over EXPLAIN and check for indexes, to avoid large scans of tables.
2. Cache static data, instead to run each time query, such as menus / counts / other widgets. (Memcache/d, other cache)
3. Avoid to use in your projects high weight CMS as Drupal, Wordpres... Have to build projects in frameworks, such as CodeIgniter, Laravel
4. Compile multiple CSS file to one large, using obfuscation and compression. for large projects use CDN, to load static content, to remove loading of the server
5. Latest php version 8.1/8.2, server with ssd, can be protected over CloudFlare, to protect from DDOS attacks, slowloris apache mod.

Share Improve this answer

Follow

answered Apr 12, 2023 at 9:46

CETb

372 ● 1 ● 4 ● 12

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.