TYPO3 Core - Bug #24318
Unnessessary serializing for memcached with variablefrontend
2010-12-10 15:43 - Ralf Strobel

| Status:   | Closed                  | Start date: | 2010-12-10 |
| Priority: | Should have             | Due date:   |             |
| Assignee: | Christian Kuhn          | % Done:     | 0%          |
| Category: | Caching                 | Estimated time: | 0.00 hour |
| Target version: | | Complexity: | | |
| TYPO3 Version: | 4.4     | Is Regression: | | |
| PHP Version: | 5.3       | Sprint Focus: | | |
| Tags:      |                        |             |             |

Description

Using the caching framework for tables like "cache_pages" in combination with memcached is typically done through the combination of:

t3lib_cache_backend_MemcachedBackend + t3lib_cache_frontend_VariableFrontend

However, this involves an unnessessary serialize/unserialize whenever variables are stored or retrieved. The "set" function of php5-memcache can handle normal variables (even objects) where database tables would have to be fed a string.

The solution would be a slightly modified version of VariableFrontend that ommits the serializing but instead just passes on the variable it is issued (as a reference, ideally). The check for string datatype in the set function MemcachedBackend would of couse also have to be removed.

Bernhard Kraft already pointed this out in a mailing list:
http://typo3.toaster-schwerin.de/typo3_dev/2010_03/msg00006.html

According to him, the serializing/unserializing can be the actual bottleneck of caching, rather than data access.

This also fits the testing results of Steffen Müller:
http://www.t3node.com/blog/testing-the-forthcoming-typo3-caching-framework-with-memcached/

Of course the storage function in php5-memcache will also have to somehow serialize the data it gets. My hope is though that this might be more efficient than doing it yourself within php.

Even if this is not the case, it might be worth thinking about ways to make nonserialized storage possible within the caching framework. There are other services such as XCache that are able to handle compiled php-code directly.

(issue imported from #M16719)

Related issues:

Related to TYPO3 Core - Bug #20582: caching of pagesections uses superfluous s... Closed 2009-06-06
Related to TYPO3 Core - Bug #24400: [Caching framework] Update to latest FLOW... Closed 2010-12-22

History

#1 - 2010-12-21 16:42 - Myroslav Holyak
According to php manual only strings and integers can be saved "as is", all else will be serialized. http://www.php.net/manual/en/memcache.set.php

#2 - 2010-12-21 23:03 - Christian Kuhn
Mmmh.

Situation:
- StringFrontend throws an exception if given data is not a string. It's a string frontend, we probably shouldn't just remove the exception.
- VariableFrontend always serializes the given data.

Assumption:
I doubt that php based serialization is much slower than serialization done in memcache (I did not benchmark!). Even if it's slower, there are php modules which speed up serialization a lot (like igbinary). And if serialization is that slow in php, this should probably handled in php upstream.

2019-08-01
Possible solutions:
1) Make sure that incoming data to the VariableFrontend is not already serialized (so no double serialization is done), use the StringFrontend if data is already serialized -> core v4 task.
2) Use igbinary as a drop-in replacement for php with an php.ini setup. Make sure the selected backend handles this -> local setup
3) Add igbinary as new caching framework frontend, make sure all backends successfully handle binary stuff -> FLOW3 commit -> backport v4 core
4) Hack some 'do-not-change-content-whatever-comes-in' frontend which could be used with the memcache backend -> FLOW3 commit -> backport v4 core.

At the moment I'm unsure which solution is best.

Links:

---

#3 - 2010-12-21 23:20 - Christian Kuhn
Here are some things we need to know to find acceptable solutions:

1) Read core code and locate positions where the caching framework with variableFrondend is feeded with already searialized data -> fix it or switch to string frontend in default setup to reduce double serialization. Benchmark if serializing a string again is really slow (for longer strings).
2) Test if igbinary does what it tells if used as a drop-in replacement for serialize()
3) Test if all backends can handle binary data produced by igbinary()
4) Benchmark igbinary in real-world solutions
5) Test a 'do-not-change-data' frontend together with memcache serialization and compare to igbinary data.
6) See if other backends can handle non serialized data nativly, too (objects, arrays), could be done with unit tests

---

#4 - 2010-12-21 23:30 - Ralf Strobel
Christian's option 3 (igbinary as new frontend) sounds like the most solid solution to me.

The function igbinary_serialize() seems to do just what serialize() does. So the rewriting should be quite minimal as well. Some testing should be done of course.

This way, the backend class could continue to demand that its input be submitted as strings.

---

#5 - 2010-12-21 23:41 - Ralf Strobel
On a related note:

It might also be a good idea to make backend_MemcachedBackend compatible with "memcached" as well (currently only supports "memcache"). Check for which is installed could be done by simply using function_exists().

Some people may want to use memcached with igbinary as default serializer. This way it would also affect serialization of session data.

I might start a separate issue for this...
True, the backend should somehow work with both memcache and memcached ... patches for this should be done in FLOW3 first. From my point of view there are currently more important tasks: We must implement the garbage collection for this backend asap ...

BTW: Currently php-memcache is broken for me in debian squeeze because delete fails due to a misleading second parameter, so I'm currently unable to do much work for this backend without much hassle in my setup. Please also keep in mind that memcache doesn't really fit the "structure" the caching framework puts into it, there are backends which handle this much smarter (like the new redis backend in 4.5 if you want an nosql solution).

If you want replace all serialize calls to igbinary_serialize(), then it's probably needed to create some t3lib_div::serialize where system will choose what method to use according to loaded php extensions etc.

FYI: igbinary support was already added to the VariableFrontend in FLOW3:
http://forge.typo3.org/issues/11443
I'll hopefully find some time to backport this to 4.5 before stable ...

That would be very nice. Just installed igbinary on my servers.

Hopefully there will eventually be a debian package as well.

The igbinary serializer in the variableFrontend will be backported from FLOW3 with issue #24400

I just noticed there already is an ApcBackend.

That of course takes me right back to where this issue started:
Unlike Memcached, APC really can store and retrieve variables without serialization. Still doing so is quite a waste of time.

Maybe, in correlation to "phpcapablebackend" there should also be an interface "nonserializedbackend". I'm sure there will be other backend storing methods that can also handle unserialized code.
Are you sure APC can store objects? Can you proof that? I ask because in this bug-discussion http://pecl.php.net/bugs/bug.php?id=8118 i have read that non-scalar values (objects, arrays) are passed via internal serialization. E.g try to search by words "[2006-07-04 23:17 UTC] rasmus at php dot net".

I'm going to run some tests myself over the next days. It's true that there seem to have been some issues in the past.... http://www.php.net/manual/en/function.apc-store.php

There it says: "It might be interesting to note that storing an object in the cache does not serialize the object".

But also: "It should be noted that apc_store appears to only store one level deep. So if you have an array of arrays, (...) it will only have the top level row of keys with nulls as the values of each key."

Such unexpectable array storing is bug and it was resolved in summer 2010 (the same link as above) http://pecl.php.net/bugs/bug.php?id=8118.

And if we want to know truth about possible serialization of objects - then there is no better way than ask developers of apc or digging in cvs/svn.

You're right. Asking one of the developers is probably the only trustworthy source.

If you haven't found other solid information so far (I haven't) I will go ahead and contact one of them.

Meanwhile, I can at least confirm that storing and retrieving cascaded arrays/objects works fine in the current version.

The variable frontend now supports the igbinary serializer and another double serialization was fixed with #20582.

I don't expect any serializer in memcache or apc to be more reliable or even quicker than the current solution.

Thus, I do not think we need to take any more actions on this topic, especially as every solution using backend capabilities would force us to create another frontend class which doesn't seem to be very useful at the moment. We should only do this if we can prove that this gives a real performance benefit. So, unless no one of you wants to test, benchmark and hack up some solution, I'll tend to close this issue within the next days.

If there is still some need to have a 'path-through' frontend together with a self-serializing backend, this should go to the issue tracker of FLOW3 anyway.

BTW: The apc backend has some serious problems which renders it unusable for most 'real-life' caches of serious size. See http://wiki.typo3.org/Caching_framework for details on this topic.
I'm still waiting for replies from the APC developers. If I could still post those here, even if the issue is closed, then I have nothing against that. I also think the solution based on igbinary sounds pretty solid.

A question I can already answer, however, comes from the wiki page you linked: "It's currently unknown what exactly happens if APC can not store additional data"

What happens is you get a PHP Warning "unable to allocate memory" and nothing gets stored. I had that a lot before upping memory size in the configuration. Now, after assigning 256 MB, I'm still far away from the limit even with several hundred pages cached. Not that I would mind a garbage collector becoming available.

Can't confirm serious memory leaks. Usage seems quite steady after a while. I'm using the newer squeeze or dotdeb packages.

Another possible solution of using APC I tried out was PhpFrontend + FileBackend.

I can only say that for me it didn't work at all. It just resulted in a lot of error messages. When I looked into the files, I didn't even find valid php code, but instead just serialized variables, wrapped in <?php ?>-Tags.

@Ralf:
Thanks for feedback on the APC backend. If a warning is raised by PHP, it should probably be caught and handled in the backend. This is actually a bug in this backend which should be tackled. We should report this to FLOW3 and see if we could come up with a unit test for this case.

It would be great if you could document your findings about the APC backend in the caching framework documentation, the documentation was just created by me and will hopefully find its way to the official documentation if all parts have been reviewed. It's a wiki page, so it would be great if you could improve the current statement.

For the memory leaks: I was able to reproduce them with native debian lenny php packages (no dotdeb) with my enetcacheanalytics extension (it has a performance suite for cache backends, check out from forge if interested).

For the fileBackend:
Do not use the PhpFrontend with the fileBackend if you are not storing PHP files. If you are caching "usual" data like strings, arrays or objects, you should combine the fileBackend with the Variable or String frontend. The PhpFrontend must be used only if storing PHP files. I have improved the documentation a bit to make a clear statement about this.

The Warning I got is discussed here: http://pecl.php.net/bugs/bug.php?id=16966

It's probably not the final behavior. They mention fixing it by having apc clear the oldest cache entries when not enough space is available, which seems pretty reasonable.

Also, if you set the ttl configuration to zero (disabled, the current default), the cache is supposed to be purged entirely once it is full. I haven't testet
this yet, however.

I updated the framework wiki documentation. Take a look if you see it fit.

---

**#21 - 2010-12-30 12:11 - Ralf Strobel**

There was still no response from the apc developer I emailed, so I went and had a look at the sourcecode myself...

The interesting function is "my_copy_zval", located here:


As it looks, apc does serialize objects, using php_var_serialize (which I guess results in the standard serialization).

However, any other datatype (numbers, strings, even arrays) is directly memcopied from the running php instance. So, as long as you are not handling objects mostly, this should be the fastest way of caching thinkable.

For arrays, this could really mean a significant edge over igbinary when loading from cache. Apc seeems to store the actual hash table of an associative array, meaning keys will not have to be re-hashed when rebuilding the content.

If I find the time, I will try to do a benchmark between igbinary+apc and just apc.

---

**#22 - 2010-12-30 19:46 - Ralf Strobel**

Well, you got to love it when test results completely disprove what you had anticipated...

I benchmarked using a multidimensional associative array of random data (integers, strings), running two different dataset sized (8kb, 8mb). Results were constant over several runs in both cases.

-------- 8 kb --------------

Loading data from uncached php file: 0.274 ms
Loading data from cached php file: 0.051 ms

- serialize() : 0.060 ms.
- unserialize() : 0.061 ms.
- igbinary_serialize() : 0.093 ms.
- igbinary_unserialize() : 0.043 ms.

- apc_store() : 0.056 ms.
- apc_fetch() : 0.047 ms.

- apc_store(serialize()) : 0.049 ms.
- unserialize(apc_fetch()) : 0.046 ms.

- apc_store(igbinary_serialize()) : 0.087 ms.
- igbinary_unserialize(apc_fetch()) : 0.037 ms.

-------- 8 mb --------------

Loading data from uncached php file: 187 ms

- serialize() : 106 ms
- unserialize() : 109 ms

- igbinary_serialize() : 221 ms
- igbinary_unserialize() : 72 ms

- apc_store() : 36007 ms
Ok, so the most obvious lesson is that \texttt{apc\_store} cannot be recommended for for large datasets, probably due to memory allocation overhead.

The second surprise for me was that \texttt{igbinary\_serialize} is actually slower than \texttt{serialize}. Since \texttt{unserialize} is faster however, I think that justifies its use in most caching environments where reads occur more frequent than writes. Quite disappointed though that the difference is this small.

Maybe most importantly: Looking at the absolute numbers, I now even doubt my original premise that serialization is a main bottleneck of caching. If 8 megabytes of complex data can be processed in roughly 0.1 seconds on a relatively weak modern server (Intel i3), then this can't really take up a lot of overall execution time, can it?

---------- EDIT ---------------

Turns out the slower serialization speed of igbinary is caused by its string compacting method (saves a bit more space in some scenarios). It can be disabled by "igbinary.compact\_strings=0" in \texttt{php.ini}.

Now the timings for 8kb are as follows:

\texttt{igbinary\_serialize()} : 0.030 milliseconds.
\texttt{igbinary\_unserialize()} : 0.044 milliseconds.
\texttt{apc\_store(igbinary\_serialize())} : 0.025 ms.
\texttt{igbinary\_unserialize(apc\_fetch())} : 0.036 ms.

It is also worth mentioning that even without compacting, the output of igbinary was always around 25\% smaller.

Note: Setting \texttt{compact\_strings=0} in \texttt{igbinary 1.0.2} gave me errors in scripts that are trying to store entire objects. I emailed the developers about it and they said it is already fixed in an upcoming version.

---

\texttt{#23 - 2011-01-18 23:11 - Christian Kuhn}

Thanks for benching Ralf!

As a sum up, APC based serialization doesn't seem to give us a real benefit which can't be done by \texttt{VariableFrontend} as well (especially since we
integrated the igbinary_serializer).

I'd like to close this issue for now, it doesn't really seem to lead to anything for now. Still, all measurements and conclusions are valid. Is this ok for you Ralf? We could still open another issue if things change ...

#24 - 2011-01-19 00:02 - Christian Kuhn

Ok, actually closing here for now. Ralf, please reopen if you have further suggestions which fit to current class logic.