

Dominique Stender

Exhausting TYPO3

Solutions for Better Performance in TYPO3 Version 4

The dmc digital media center GmbH runs e-commerce shops based on TYPO3 with several million page requests and over 100 gigabytes of daily traffic. In order to achieve a seamless operation with a reasonable amount of hardware, central components of TYPO3 were expanded or replaced. In this article we present some of these solutions and indicate where the potential for further improvements in TYPO3 performance might lie.

A shop platform was developed on a TYPO3 base for clients in the e-commerce sector. It uses the strengths of TYPO3 as a content management system to expand the typical components of an online shop and connect these to various backend systems (ERP systems, external servers). TYPO3 is only naturally fast when a page is completely capable of being placed in a cache. However, in the field of e-commerce, with dynamic shopping cart symbols and other user-dependent elements, this is difficult to impossible.

Faster websites bring higher turnovers

This article presents several ad hoc measures which achieve a high throughput and hence minimal waiting times, even with pages that are only partially stored in a cache. First, the speedy delivery of websites enhances customer satisfaction and thereby increases the potential turnover of the shop. Second, it is not long before the web servers are 'free' again and able to process other customer requests. After all, when several hundred users are simultaneously surfing the system and six and seven figure turnovers are generated each day, every millisecond counts.

Scalable hardware landscapes

So that the high number of page views can be processed, the delivery of the pages must be distributed across several web servers. A load balancer has proved useful: it takes up every outside request and passes it on to a cluster of web servers. In addition, the load balancer adopts the SSL encryptions of sensitive content, for example during the ordering process. Since the load balancer accomplishes this with a hardware module, the encryption is noticeably faster and takes pressure off the web server.

Each web server is an independent machine (often an IBM Blade) with at least two CPU cores and two gigabytes of RAM. All web servers are linked to one another and to the load balancer through a gigabit network.

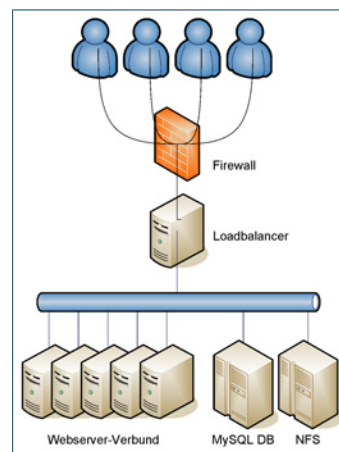
The TYPO3 installation and all the files belonging to it are gathered in their own server which makes these files globally available through a network file system (NFS). Thus all web servers work with identical files. The modification of a file on a web server is immediately recognised by all other web servers.

Marketing measures and the like may result in peak demands. Should demand peak in such a way that the existing web servers cannot do a satisfactory job, the architecture described above makes possible the quick and relatively inexpensive integration of further web servers into the cluster so that the system's total throughput can be increased. The potential loss of turnover in systems of this magnitude will outstrip even the price of several web servers.

The content of the database is found in a central server. The hardware roughly corresponds to that of a normal web server but, according to specification, has up to four CPU cores, eight gigabytes RAM, as well as larger and faster hard drives in the RAID

network. All systems which are necessary for the operation (load balancers, NFS servers etc) appear twice. Should one load balancer cut out due to a hardware defect, the second immediately takes over the requests and the whole system stays online.

Total performance is a complex mechanism



In large web projects the use of a load balancer, a web server network as well as a separator server for MySQL and file systems has proved useful.

In order to attain a scalable total system, all the components must work together seamlessly and be optimally configured. The most important components are the distributed file system such as the NFS, the database (my.cnf), the web server (httpd.conf), the configurations of the software such as TYPO3, the php.ini, as well as the clean and efficient programming of TYPO3 itself, of extensions and of other components.

Since many of the individual components are tightly intermeshed, it is in the nature of the matter that the cause of a bottleneck is not always obvious.

Take a performance bottleneck in the file system. It results in every „require()“ function taking a few milliseconds longer – and this adds up. A symptom is an increased CPU load on the web servers, which has nothing to do with the actual cause.

A scaling problem in the database

As described above, it is easily possible to integrate additional web servers into the system. It is not quite so simple with the MySQL database because all the web servers fall back on the same database server. After a certain number of requests, MySQL on standard hardware loses scalability. Adding more RAM and more CPUs will not help. Thus, right from the start, TYPO3 should not be put to work with persistent database connections. These may be reached via: „\$TYPO3_CONF_VARS[‘SYS’][‘no_pconnect’] = 1;“.

Although PHP constantly builds new connections, these disappear from the connection pool of the database with the eradication of the scripts. Therefore MySQL has to administer fewer connections than would be the case with persistent connections. The web servers can easily manage the additional requests which develop on their side.

Recently several solutions have appeared through replication mechanisms (so-called „read only slave“ or „master master“ configurations) and through the MySQL cluster engine (NDB), which attempt to solve the availability and scaling problems on the side

of the database. However, in practice it quickly becomes apparent that both solutions have disadvantages which would make their adoption by a web system appear questionable.

Thus the best solution for a TYPO3 system is to reduce the number of requests to the database and to thereby not allow the problem to arise in the first place. When a page can be made completely into a cache, this should be the first measure taken.

ext:dmc_highPerformance

dmc digital media center GmbH has bundled many of the improvements made to TYPO3s performance into the extension „dmc_highPerformance“ [1]. This extension is available in the extension repository and requires the pear packet „PEAR::Cache“ [2] to be in the server. Currently the extension supports TYPO3 versions 4.0.5 to 4.0.7 and 4.1.0 to 4.1.2.

The extension „dmc_highPerformance“ improves the performance of TYPO3 in various areas, which are described in the following. The accompanying documentation should on all accounts be read before installation, so that no undesirable side effects manifest themselves.

Unnecessary database traffic

Even where pages are wholly or partially stored in a cache, TYPO3 is forced to send a relatively large number of queries to the database. The files most often queried are from the tables „sys_template“ and „pages“. Although the files in these tables tend to be static, TYPO3 still has to check with each page view, for example, whether the pages indicated in a menu are visible under the current circumstances or not. As a result, in a heavily frequented system, the database is constantly bombarded with the same queries.

Thus, as necessary as the tenders to the tables „pages“ and „sys_template“ are, in most systems there is no reason not to store the sum of all TypoScript-Templates or the availabilities of the TYPO3 pages in a cache for a short time, and hence reduce the load on the database. Even 60 seconds of cache validity spares the database from several hundred requests per minute and make the total system noticeably faster. The question remains: where do the caches go?

Additional caching through PEAR::Cache

The module „PEAR::Cache“ is able, as desired, to temporarily save structured files. For performance reasons the caching is offered as files. After all, pressure is supposed to be taken off the database. In doing so, files should not be stored in networked file systems because these are slower than local file systems. To be sure, every web server will then have to administer its own cache, which will increase the number of database queries in the short term. However, the gain in speed offsets this disadvantage.

Now the challenge is to expand the core of TYPO3 in such a way that files are cached in the right places through PEAR::Cache.

Caching of database queries

The most wide-ranging expansion to which the TYPO3 core was subjected concerns the caching of database results described immediately above. The PEAR::Cache module is used to cache MySQL results as PHP arrays in cache files which are situated in the local file system. The reading and preparation of a PEAR::Cache file takes about as long as a database query, so long as the database is not overloaded. Thus local caching can dramatically reduce the load on the database.

For this reason the „highPerformance“ class of „dmc_highPerformance“ has a group of methods which allows SQL statements to be cached without much modification to the TYPO3 core. Although many classes and methods still have to be rewritten with

the XCLASS logic, the changes are clearly visible and a comparison between the original method and the overwritten one is quickly accomplished when updating to a more recent version of TYPO3. The attachment of a member of the PEAR::Cache class as well as the generation of the cache IDs take place transparently within the „highPerformance“ class.

If the frontend cache is deleted via the TYPO3 backend, the PEAR::Cache files must be deleted from the web servers too. This is achieved with a hook. So that this works seamlessly, however, all the internal host names of the web server must be configured in a cluster in the extension. It must also be possible to communicate between the web servers of a cluster via HTTP.

Making better use of the MySQL query cache

MySQL saves frequently dealt with statements in the query cache together with their result in the main storage, so long as the query cache has been activated [3]. Here it is important that only statements which are in fact identical are selected from the cache. A small change in the WHERE clause, for example, leads MySQL to ignore the query cache and to reapply the statement.

In its database tables TYPO3 often works with time stamps in order to determine when a page, page element or TypoScript template should go online or offline. In the same way, cache validity and other matters are administered through timestamps which are accurate to the second.

Although nothing is wrong with it as such, a SQL statement like „...WHERE pages.starttime<=\$GLOBALS[‘SIM_EXEC_TIME’]...“ can result in the MySQL query cache becoming invalid as seconds tick by. Here the question arises whether, for the sake of the greater efficiency of the query cache and hence an overall improvement in performance, the online and offline states really need to be accurate to the second. Accuracy to within ten minutes should suffice and, in addition, improves the effectiveness of the MySQL query cache.

Thus, so that „roundTime()“ is used for the rounding of timestamps, the static method of „highPerformance::roundTime()“ was implemented in the extension and relevant parts of the TYPO3 core were rewritten. This method has an optional second parameter with which a rounding factor of ten minutes can be customized for individual cases.

REPLACE instead of DELETE and INSERT

For some tables, especially for the heavily frequented tables for the caches in the frontend sessions, TYPO3 uses a combination of a DELETE and an additional INSERT statement in order to update contents. Briefly, in the case of front-end caches, the reason is as follows. If a page which can be, but has not yet been, cached is selected, TYPO3 makes a provisional entry into the cache table. From that point onwards, visitors to the page see the announcement, „Page is being generated.“ Meanwhile the often time-consuming generation of the HTML code continues. Finally the temporary entry into the database is deleted via DELETE, and the HTML which has just been generated is written into the table via INSERT. From then on the page has been cached, and future visitors will see the content which TYPO3 has just written into the database.

The problem: between the DELETE and INSERT statements a little time passes. In large systems this can lead to a so-called „race condition“ where processes running in parallel get in the way of one another. As a result there is a high probability that the page „Page is being generated“ does not disappear.

Luckily, MySQL has a REPLACE statement. It behaves just like an UPDATE statement when the dataset, which has been defined by the WHERE clause, already exists. If that is not the case, REPLACE

behaves just like INSERT. Thus only a single statement is sent to the database with the result that race conditions are made impossible.

The extension „dmc_highPerformance“ integrates a method by the name of „t3lib_db::exec_REPLACEquery()“ into the database layer of TYPO3 and replaces the former DELETE/INSERT logic where necessary. The disadvantage: for all incoming queries the page is completely rendered until this process has come to an end and the cache content has been written into the database.

REPLACE is not a part of the SQL-ANSI standards and hence is not necessarily available for other database types. An alternative to the DELETE/INSERT combination is the UPDATE/INSERT combination. With this combination the attempt is made to update a dataset, regardless of whether it already exists or not. If the UPDATE fails because no dataset could be found, the dataset can be added through INSERT. However, two disadvantages of this combination when compared to DELETE/INSERT are that (1) the UPDATE statement will more likely than not produce an error which will be logged; and (2) two statements are still needed rather than one, and this costs time.

Optimising the MySQL master-slave replication

Part of the solution for a highly potent system is a master-slave replication of the database. The MySQL database will be running on two servers. One machine – the master – is known to TYPO3. All queries, whether reading or writing, will be directed to the master and be answered by it.

The second machine – the slave – is not used actively. Instead, it serves as backup in case the master should cut out due to a hardware defect. To achieve this, a so-called „bin log“ is activated on the master. The MySQL master writes all the SQL statements changed by the database into a binary log file. The slaves then read this bin log out and execute the statements in their local tables. In this way the state of the master files is replicated by the slaves almost in real time. Should the master then cut out, the files are still present and the slave can be reconfigured in such a way as to make it into a master. Details for this procedure may be found in the MySQL manual [4].

Now it does not necessarily make sense for the slaves to replicate every statement that TYPO3 sends to the database because the writing of the bin logs places an additional burden on the MySQL machine. The cache tables and ultimately the frontend sessions need not necessarily be secured. So long as capacity permits, the replication of these files is advantageous; but in many systems it is not truly necessary.

Unfortunately MySQL does not allow the bin log to be configured in such a way that individual tables can be exempted from replication, only individual databases may be excluded. That said, where the MySQL user has SUPER privileges it is possible to switch the replication on and off for a short time within a database connection with a SQL statement.

The solution: using the static method „highPerformance::switchMySQLBinLog()“ to check whether the MySQL bin log is currently active and has subsequently been deactivated as desired. A renewed selection of the method switches the bin log back on. The methods of the TYPO3 core relevant for this were overwritten in such a way as to deactivate the bin log before it can deposit a writing statement, to the cache tables for example, and finally reactivate them. The optimisation of the bin log must be activated via the PHP constant „MYSQL_BINLOG_SPEEDUP“.

Conclusion

Off the shelf, TYPO3 has a relatively hard time with high performance: one notices that its creation took many years. Some included functions which promise performance gains can, if at all, be applied to web servers run by clusters only through customisation, such as HTML caching in local files via „\$TYPO3_CONF_VARS[FE]pageCacheToExternalFiles“. In particular, the strong dependence on the database results in something amounting to a dead end.

Nevertheless, TYPO3 Version 4.0 can be applied with some effort to large, dynamic web systems. The extension introduced here, „dmc_highPerformance“ can reduce this effort, since it bundles important measures for improved performance. Meanwhile, the development of the completely new TYPO3 Version 5.0 proceeds apace. It will be interesting to see where TYPO3 goes with respect to performance matters.

Links and Literature

 [Softlink 1847](#)

- [1] dmc_highPerformance: <http://typo3.org/extensions/>
- [2] PEAR::Cache: <http://pear.php.net/package/Cache>
- [3] MySQL Query-Cache: <http://dev.mysql.com/doc/refman/5.0/en/query-cache-configuration.html>
- [4] MySQL-Replication: <http://dev.mysql.com/doc/refman/5.0/en/replication.html>

THE AUTHOR



Dominique Sender has been working for several years in the online sector for the dmc digital media center GmbH as an IT consultant and software architect. His main responsibilities are the development and maintenance of in-house solutions for e-commerce websites as well as giving technical advice to clients.