Optimization of back-end WEB server

K Zhigalov$^{1,2,5}$, M V Chirikova$^3$ and I M Daudov$^4$

1 V.A. Trapeznikov Institute of Control Sciences of Russian Academy of Sciences, Moscow, Russia
2 Kh. Ibragimov Complex Institute of the Russian Academy of Sciences (CI RAS), Russia
3 Irkutsk National Research Technical University, 83, Lermontov Street, Irkutsk, 664074, Russia
4 Chechen State University, 17a, Dudaev Boulevard, Groznyy, 364015, Russia
5 kshakalov@mail.ru

Abstract. The application bottleneck is the database architecture. The applications developed more than 10 years ago suffer from low database performance. The databases in companies strongly enlarged during this time and began to work quite slowly. The article describes how the inclusion of master-master replication will optimize the backend operation to some extent without resorting to major changes. It will allow even distribution of requests across several (in this case, two) servers, regardless of the requested method.

1. Introduction

The most extensive optimization possibilities of application can be obtained due to the possibility of SSI. SSI is server side inclusion. In fact, SSI blocks are simple html comments containing a prescription on how a server should handle them. This technology allows submitting an html template to nginx with ssi blocks placed in it, the contents of which nginx will receive from other places.

2. Creation of processor stacks

If at the same time we get rid of some elements that are unique to the user on the common content (for example, numbers in task lists, indicating the number of new messages), we can create a shared cache of lists and tasks between users. The tasks in this case will be very convenient to cache if we use cache flush.

If we add cookies, which will store the user's role in the system and information about access to projects, it will be possible to use a cache list shared between certain user groups. For example, a group of managers sees the same lists, but on the pages with them there is information unique to the user.

Let us consider stage by stage, what should be done so that the described becomes possible:

1. The preparation of a common template that will be the same for all users. It can be cached as we like, or it can even be a static html file that nginx will give from the directory with the application files.

2. The preparation of the index.php script (the main and only entry point into the PMC application) so that we can get separate blocks of pages that will not contain unique content, and not the entire page as it is now. These can be messages inside tasks or lists of tasks, selected and sorted according to certain criteria. We will also need to create the ability to separately request user-unique content for the
corresponding common block.

3. The preparation of a script that, will receive unique content and fill in the necessary places with it after loading the page with a separate request. This is various metadata, reflecting the number of new messages, unread messages, messages from contractors, which are marked in a special way. There is no problem filling it after loading the list by creating empty elements with corresponding identifiers in the html list template.

4. The configuration of nginx to work with ssi blocks using directives in the context of the ssi on server; and ssi_types if types other than text / html will be used in ssi blocks

To include dynamic content in the ssi block, use the `<! - # include virtual = page_uri -> construct, to include a static file - <! - # include file = file_path ->`

The description may not give clear understanding how it works. It is much better to illustrate the operation of this mechanism through the example.

In order to apply the technologies described in this and subsequent sections, a lengthy and costly modification of the application is required. To demonstrate the operation of technology, we can use a test bench.

**3. Creation of test application**
A separate stack for the application will be created using Docker Compose. Nginx will render one page - a simple html template created on the basis of one of the task lists of a real application. Instead of the main content of the page, the ssi block will be included in it. The contents of the block will be generated by a php script, for which a separate container with the Apache web server and php will be started. It will receive two elements from the mysql base - task 1 and task 2. Each element will consist of two columns: name and time. Php will enter the current date and time in the “time” column. It is necessary to identify whether a block has been taken from the cache or received from the backend. We can enter the task by clicking on the name and see a list of several messages also received from the database. Messages can be added by sending a POST request to the page with an open task.

During the input of information to messages, it will be written to the app database. The structure of its tables in this case is not important and will not be described [4].

Please note that in the figure there are two DBMS servers. Master-master replication is configured between them. At the moment, the application can use a common alias for two servers to resolve the name of the SQL server, but for simplicity, mysql1 is used at this stage.
The application will have only two types of pages. The main page displays the task headers:

![Image](image1)

**Figure 2.** Main page of test app.

We can click each of the headings in the name column and get on the page with messages and the form for sending them:

![Image](image2)

**Figure 3.** Page with messages.

Accordingly, we can write a message to the page entering it in the text box and clicking the Enter button.

The nginx configuration file used in this example is shown in Appendix 8. The setting allows caching the contents of the table with tasks for a short time, but it prevents the contents of the tasks from getting to the cache.

The static index.html file includes several ssi blocks, the choice of which depends on the show argument. The full text of the file is given in Appendix 6.

Depending on the show argument, the render.php script passes the arguments which form the page.
In order to cache index.html template, we need to make small changes to the configuration file - add the map directive which defines the $bypass variable and the use of the cache for location / (the configuration file in the application already contains these changes).

This example shows that:
1. In order to unload the back-end of the Apache server, caching of one of the types of dynamic pages is used. This application significantly speeds up the application and part of the load from the back-end server in the considered case in the main part of the work;
2. The back-end server no longer needs to generate the main page template each time. This takes an insignificant amount of time for php, but taking into account that this happened absolutely every time when accessing any page, the gain can be significant
3. After the html template begins to be cached, there is no need to get it from the back-end server. This is quite significant, since the operation is also performed every time a user requests a page.

For reference, the application is available at http://ssi.pictcut.com:8481

4. Performance analysis after the changes
No one really uses the test application. During the collection of data and in the following sections we will use the following pattern that describes the user's movement on the site.

At the beginning, the server cache will be completely cleared:
1) http://ssi.pictcut.com:8481/index.html?arg=table – loading the task list page from the back end, as there is no cache yet;
2) http://ssi.pictcut.com:8481/index.html?show=messages&num=1 – transition to the first task. It is not cached on nginx, but there is also a DB cache in which everything that can be cached on the page will go to the cache since after executing the request the application will direct the user to the same page;
3) http://ssi.pictcut.com:8481/index.html?show=messages&num=1 – POST request, adding the mail. After adding a message, the database cache is invalidated and the subsequent selection is obtained from the database again. Everything that can be cached on the page will go to the cache since after executing the request the application will direct the user to the same page;
4) http://ssi.pictcut.com:8481/index.html?arg=table – outdated page from the cache will be returned and a background update will be performed;
5) http://ssi.pictcut.com:8481/index.html?show=messages&num=2 – opening the page with task 2;
6) http://ssi.pictcut.com:8481/index.html?show=messages&num=2 – adding the message to task 2;
7) http://ssi.pictcut.com:8481/index.html?arg=table – return to main page.

The last 3 steps will be performed to collect additional data. To illustrate the work of the cache, lines 14 and 30 are added to the render.php script:

echo sleep(1);

It will slow down the script for a second, otherwise the pages load extremely fast due to their simplicity.

Thus, the access log will be as follows:

![Figure 4. Access Log of Test app1.](image)

As it can be seen, for the first time the main page is not in the cache and is obtained from the back-end server in ~ 1 second. POST requests are completed in ~ 20 milliseconds. The time indicated where
the cache status is STALE is the time it took the server to get the updated page from the backend.

Accordingly, in this case, for 7 page visits, caching saved about 2 seconds. In the case of the present application considered in the work, the list is the most frequently used page.

5. Failure stability

High availability of systems and their resistance to failures of individual nodes is an extremely important quality of web applications [2, 3]. Load balancers allow distributing client connections between multiple back-end servers and monitor their status. This section will analyze failure stability of the Apache running application with the nginx balancer.

Testing will be carried out with a “flushed” cache, that is, filled with cached pages (in fact, only one page is cached for this section). In this and the following sections on stability testing, the testing procedure will consist of the following actions:

1. For this example, the container with Apache will be turned off, for subsequent examples with PHP-FPM. Then try to load the main page, and then try to go to the task page.
2. If a page with messages and a form is displayed, try to leave a message;
3. If after trying to leave a message the page has loaded, try to return to the main page with a list of messages.

Thus, for this example of the application, we need to move to the stack folder where the docker-compose.yml file is located and run the sudo docker-compose pause apache command.

After that, nginx loaded the main page with a list from the cache, but when we tried to access the tasks page after waiting, we received a 504 Gateway timeout response, which meant that the timeout from the back-end server was exceeded. In the error log, an entry appeared:

```
2019/10/06 17:23:55 [error] 385#385: *2829 upstream timed out (110: Connection timed out)
while reading response header from upstream, client: 188.65.209.195, server: ssi.abelov.com, request: "GET /index.html?show=messages&num=1 HTTP/1.1", upstream: "http://192.168.16.55:8188/index.html?show=messages&num=1", host: "ssi.pictcut.com:8481", referrer: http://ssi.pictcut.com:8481/index.html?arg=table
```

When we tried to access the list page again, an error 504 was also received, since more than a minute had passed and the page was deleted from the cache.

6. Conclusion

We can conclude that with this architecture, nginx practically does not improve the availability of the application in the case of a failure at the back end. The only way to increase it is to deploy additional containers for the Apache layer.

References

[1] Yegorova E V, Klyuev R V, Bosikov I I and Tsidaev B. S 2018 Evaluation of use of effective technologies for increasing sustainable development of natural and technical system of oil and gas complex Sustainable Development of Mountain Territories 10(3) 392–403
[2] Zhukovskiy Y L, Koteleva N I 2019 Electrical equipment maintenance system with elements of augmented reality technology *IOP Conference Series: Materials Science and Engineering* 643(1) 012024

[3] Zhukovskiy Y, Batueva D, Buldysko A, & Shabalov M 2019 Motivation towards energy saving by means of IoT personal energy manager platform *Journal of Physics: Conference Series* 1333(6) 062033

[4] Urazbakhtin R Yu, Yungmeyster D A 2019 The results of studies of the tunneling rescue complex for coal mines *IOP Conf. Ser.: Mater. Sci. Eng.* 560 012130