**Main results from test runs**

**Capacity testing**

|  |  |
| --- | --- |
| Number of anonymous users | 50 |
| Number of editor users | 1 |
| Number of admin users | 1 |
| Ramp-Up-Period (in sec) | 1500 sec |
| Duration | 30 min |

**Results**

**System is very unstable to increasing load. It was very hard to complete capacity testing, because reaching of the saturation point breaks the application. CPU using on server machine become 100% and it stops responding. Therefore, a couple of tests have been executed and results have been compared.**

**25 transactions per second was defined as a saturation point, what corresponds 40 virtual users.**

**Load testing**

|  |  |
| --- | --- |
| Number of anonymous users | 16(45 %), 25(65 %), 32(85 %) |
| Number of editor users | 1 |
| Number of admin users | 1 |
| Ramp-Up-Period (in sec) | 180 sec |
| Duration | 30 min |

**Results**

The system under 45 percent of the maximum load operates without errors and without increasing processing time. An Internal Server Error was detected several times when the load was increased to 65. At a load increase of 85 percent, the number of errors also increased, and the system failed once with CPU overload, which happens from time to time when the load is increased. There were no deviations in the use of memory or CPU other than those that could be explained by the work of external services of the environment.

**Stress testing**

|  |  |
| --- | --- |
| Number of anonymous users | 50 |
| Number of editor users | 1 |
| Number of admin users | 1 |
| Ramp-Up-Period (in sec) | 500 sec |
| Duration | 30 min |

**Results**

Stress testing was hindered by the low fault tolerance of the application, and when the load exceeds 40 users, the application most often stops responding. CPU usage goes down to 100. Unfortunately, it is not yet possible to find a specific reason for this.

**Scalability testing**

|  |  |
| --- | --- |
| Number of anonymous users | 16 |
| Number of editor users | 1 |
| Number of admin users | 1 |
| Ramp-Up-Period | 180 sec |
| Duration | 30 min |

|  |  |
| --- | --- |
| **Memory** | **Cores** |
| **1 Gb** | **6 CPUs** |
| **2 Gb** | **6 CPUs** |
| **3 Gb** | **6 CPUs** |
| **4 Gb** | **6 CPUs** |
| **5 Gb** | **6 CPUs** |
| **6 Gb** | **6 CPUs** |
| **6 Gb** | **1 CPUs** |
| **6 Gb** | **2 CPUs** |
| **6 Gb** | **3 CPUs** |
| **6 Gb** | **4 CPUs** |
| **6 Gb** | **5 CPU** |

**Results**

We can confidently say that an application is unable to handle the load having a single core. The configuration with 2 cores and 6 gigabytes of RAM proved to be the best. Increasing the number of cores will lead to long processing time delays and increasing RAM will lead to lower fault tolerance.

**Stability testing**

|  |  |
| --- | --- |
| Number of anonymous users | 16 |
| Number of editor users | 1 |
| Number of admin users | 1 |
| Ramp-Up-Period | 2.5 minutes |
| Duration | 12 hours |

**Results**

**With the optimal configuration, the system has been able to operate stably for 5 hours. However, there was another CPU jump at the end and the application stopped responding. Soon it turned out that during the test the number of context toggles increased much more than the 30K recommended for two cores.**

**I found that all the time after the crash, the system handles 10 requests, in 10 threads with 10 connections. In addition, I found that the specific system call that is running all this time is ntdll!RtlUserThreadStart, which as I noticed after reading the forums there are many problems associated with high CPU**

**I couldn't get a superficial look at what caused the problem, but it seems to me that the problem is exactly that the application is running on a virtual machine, which means that the cause of the failure may be the peculiarities of the virtual machine's interaction with cores. Looking at memory usage as well as disk operations, I can assume that the failures are not related to them, because you can't see catastrophic degradation on the charts, and even after I've experimented, I've come to the point where failures occur regardless of the flow scenario and the load used, the load only affects how fast it happens.**

**Volume testing**

|  |  |
| --- | --- |
| Number of anonymous users | 16 (45 % from maximum load) |
| Number of editor users | 1 |
| Number of admin users | 1 |
| Ramp-Up-Period | 2.5 minutes |
| Duration | 30 minutes |

|  |  |
| --- | --- |
| **1** | **100 Posts** |
| **2** | **1000 Posts** |
| **3** | **2000 Posts** |
| **4** | **5000 Posts** |
| **5** | **1000 Posts with a large text and 1000 with 1MB image** |

**Also complete and analyse the same testing for application with DB data source. DB that I was using is SQLite. Posts were generated by the same script as for File System testing. Capacity testing showed the same capacity point as for FS configuration, so the number of users that was taken for tests is the same.**

**Results**

The results of the file system tests do not contradict expectations: we have a directly proportional relationship of response time to the number of records in the application.

Adding images to the posts has not changed the speed of processing the request, most likely because the generator receives the response and information about static resources as well as the image, but does not request them to the server.

Also, as the number of posts increases, the probability of receiving a 500 response from the server increases.

**Changing the server configuration to use a database instead of a file system has resulted in performance degradation although it has significantly increased application resilience, which was revealed by stress and capacity tests. Problems with using the database include the fact that Non HTTP response message: invalid code lengths set errors have started to appear regularly. The connection of these errors with the database still needs to be clarified.**

**I'm sure that the performance when working with a database depends heavily on the database itself. SQLite showed bad results compared to the use of the file system. Given the comparison of the median and 90 percentiles, it is worth to conduct tests with an even greater number of posts, maybe further use of the database becomes faster than the file system.**

**Maximum data size needs to be found with more tests.**

**Detailed Analysis**

The main issues that should be addressed in the first place are not the application issues, but the environment and database that the application is working with.

If the system stops responding to queries due to the intricacies of the virtual machine, another server could be used and there would be no such problems, and the application would have much more capacity. In addition, during the tests it was observed that different services can use quite a lot of resources and together with the load created by the generator used a critical percentage of CPU.

Behind the results of the volumetric test, you could say that the file system is more suitable for storing data, but you keep it that way. The database that was being used was not the most productive of those that could be used, but it would have taken longer to connect any other database. Also, there might not have been any errors under the other database that appeared when it was connected.

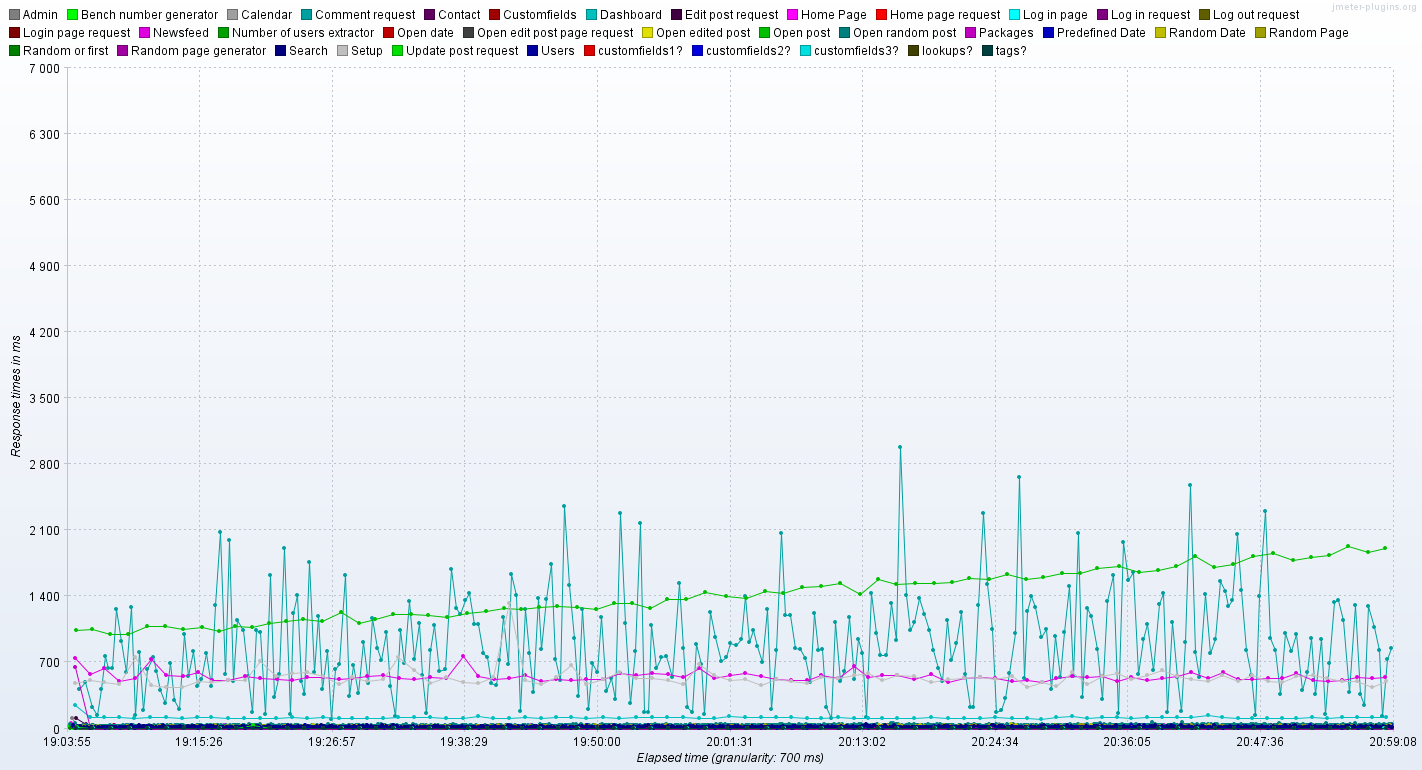
The amount of test data that was used in the bulk testing was not critical, it is worth checking a lot of posts and with a lot of text and style. Then the difference in performance between the database and the file system will be clearer.

These are the first steps that should be taken to improve the application performance.

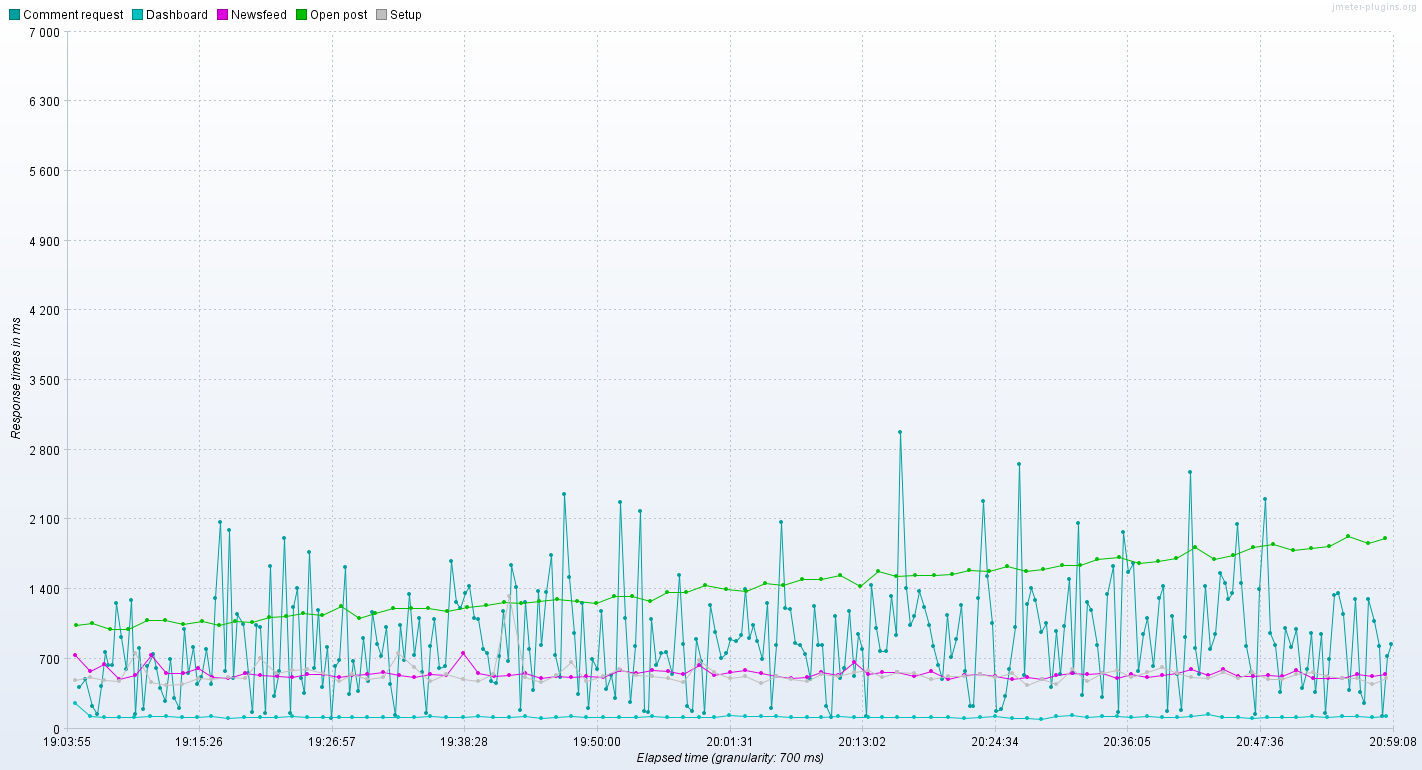
* Find an operating system that will not crash when you are running Virtual Windows 10.
* Connect to the service a productive database, which is available for the operating system.

To the question of the performance of the application itself

Here is a graph of the answers to all requests during a two-hour stress test with a 45% of capacity load:



Among these, we can find and highlight the requests that take up most of the time.



As expected, the most productive requests were those related to reading and editing of posts. This is opening a post, commenting, and 3 requests that are used on the administration page, they request information about the total number of posts, their status and properties.

In general, it seems to me that judging by the increase in the opening time of the post, the load model is not quite suitable, users will not make so many comments. In this case, such an increase in the opening of the post would not be observed.

Here is some information about slowest posts

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Transaction | Avg | Median | 90% | 95% | 99% | Max |
| Packages | 12 | 4 | 7 | 11 | 20 | 4472 |
| Open post | 1414 | 1396 | 1786 | 1872 | 2001 | 2135 |
| Dashboard | 119 | 113 | 135 | 145 | 180 | 1028 |
| Setup | 542 | 518 | 634 | 669 | 1339 | 6374 |
| Newsfeed | 552 | 537 | 631 | 667 | 885 | 2317 |
| Comment request | 1060 | 347 | 3519 | 4314 | 5161 | 6096 |

Packages is also being used on Administration page, it is fast, but has one of the biggest max.

Anyway, all requests except comment for 99% of users will be processed in less then 2 seconds, when compared to the user-friendly 3-4 seconds, that's very good.

It is worth noting that the heaviest request is to comment the post, and when optimizing it, start with it. Also pay attention to the packet post, the reason that gives so much delay may be related to other slow posts.

All the queries that had errors here, too, are comments and dashboard queries. Since they are write and read from disk, I think that the problem is with disk operations and queues.

**Conclusion**

In conclusion, I would like to say that the application by modern standards is quite productive and most likely, having solved the problem with server failure, could have withstood the load and more. Overall, there are far fewer problems than there could have been, and the paths to solving those that are quite clear. Thank you.