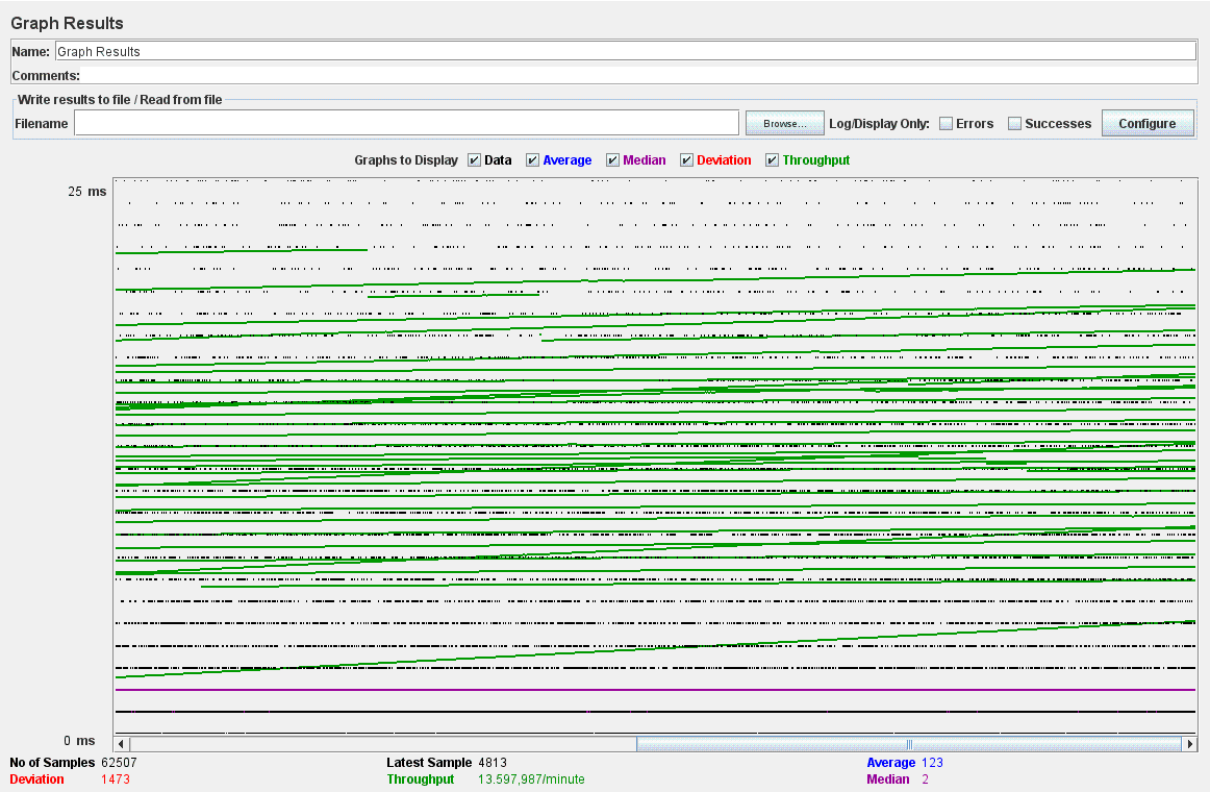
All the tests were done on Windows 10, server was hosted on a non dedicated IIS on a system with Intel Skylake i7 6700k (4.2 GHz, quad core in 8 logical processors), 16 GB available RAM (server was using around 250 MB).

Load test performed with jMeter, at 100 users with 5 seconds of Ramp-Up period, 2x loop.

In test, login is executed, then 1 MB large file is downloaded, after that a new folder is created, opened and deleted, then user logs off.

Speed was fast because of optimizations of webpage loading and server cache.

While the theoretical throughput is measured at around 13 thousand request per minute the real estimation is around 500 if users perform heavy operations.



Bottlenecks appeared in front end, especially in generation of CSRF tokens. In case of larger load the expected bottleneck would be in a database.

From controllers, the most loaded one was MainController, which is managing all of the logics regarding folders and files thus doing the most work.

Profiling was done with JetBrains dotTrace Performance Profiler.

User code hotspots

6.695 ms _Page_Views_Shared__Popups_cshtml.Execute
2.741 ms QuickPulseServiceClient.SendRequest
2.562 ms _Page_Views_Shared__Layout_cshtml.Execute
725 ms Microsoft.ApplicationInsights.Web.Implementation.WebTelemetryInitializerBase.OnInitializeTelemetry (2)
700 ms SyntheticUserAgentTelemetryInitializer.OnInitializeTelemetry
683 ms ApplicationDbContext.Create
480 ms System.Runtime.CompilerServices.IAsyncStateMachine.MoveNext (7)
176 ms IdentityFactoryMiddleware+<Invoke>d__0`2.MoveNext

Function Name	Time, ms	Own Time, ms
Microsoft	155.865	810
Microsoft.ApplicationInsights	135.252	69
Microsoft.Owin	13.938	386
Microsoft.AspNet	3.517	160
Microsoft.VisualStudio	3.157	195
ASP	18.180	88
ASP._Page_Views_Shared__Layout_cshtml	10.417	22
ASP._Page_Views_Shared__Popups_cshtml	7.258	0
ASP._Page_Views_Shared__LoginPartial_cshtml	194	0
ASP._Page_Views_Home_Index_cshtml	175	0
ASP._Page_Views_Shared__Errors_cshtml	124	66
ASP._Page_Views__ViewStart_cshtml	14	0
OnlineFileSystem	1.219	61
OnlineFileSystem.Models	879	0
OnlineFileSystem.Models.ApplicationDbContext	776	0
OnlineFileSystem.Models.AuthorizationFilter	103	0
OnlineFileSystem.Controllers	166	0
OnlineFileSystem.Controllers.MainController	154	0
OnlineFileSystem.Controllers.FrontController	12	0
OnlineFileSystem.ApplicationUserManager	102	0
OnlineFileSystem.ApplicationSignInManager	72	61
System	36	12

Webpage loading is optimized via cache (already integrated into ASP.NET and IIS server) and via minification of files send from server to client.

In code the database read is avoided if that is possible. For searching and selecting a primary keys are used (if that is possible). The biggest part of application is optimized via lazy loading as file content will be loaded only on demand. So file content (the biggest table in a database) will be loaded only when we are uploading, downloading or deleting a file and the table will never be read as a whole (always searched via primary key).