



Webalkalmazások teljesítményvizsgálata és skálázhatósága

Szakdolgozat

Kapui Ákos (B3V4S1)

Konzulens: Micskei Zoltán

BME Méréstechnika és Információs Rendszerek Tanszék
Informatikai technológiák szakirány, Rendszertervezés ágazat

Összefoglaló

A Web robbanásszerűen növekvő népszerűsége miatt a webalkalmazások is egyre növekvő látogatószámnak vannak kitéve. A növekvő látogatói szám növekvő terhelést is jelent, amire sok esetben az egyes alkalmazások nincsenek felkészítve. Ilyenkor ideig-óráig megoldást jelenthet egyre erősebb hardver használata, de egy idő után biztos beleütközünk valamilyen korlátba, ami megakadályozza a további skálázhatóságot. A szakdolgozatom célja éppen ezért annak megvizsgálása, hogy milyen módszerekkel és eszközökkel lehet webes alkalmazások teljesítményét mérni és javítani.

A szakdolgozatom első két fejezetében áttekintem azokat a háttérismereteket, amik teljesítményméréshez, teljesítményoptimalizáláshoz és skálázható webalkalmazás készítéséhez szükségesek. Kitérek a már jól bevált teljesítményoptimalizáló módszerekre, és bemutatok különféle eszközöket, amik megkönnyítik a mérést, a tesztelést vagy akár az eredmények elemzését.

A harmadik fejezetben három különböző webalkalmazás mérésének környezetét, illetve optimalizálásának folyamatát tervezem meg. A webalkalmazásokat és felhasznált szoftvereket egyaránt úgy választottam meg, hogy legyen közte, egyszerű és komoly, illetve fizetős és nyílt forráskódú is. A három különböző mérést három különböző szempont szerint állítottam össze, attól függően, hogy az optimalizálást egy felhasználó, egy üzemeltető vagy egy fejlesztő végzi.

A negyedik fejezetben mindhárom szempont esetén bemutatom a mérések folyamatát, fontosabb lépéseit és a mérések során szerzett eredményeket. Az egyes eseteket megpróbálom úgy bemutatni, hogy az más webalkalmazás mérésére is könnyen alkalmazható legyen, és a tervezés során leírtakhoz illeszkedjen.

Végül az ötödik fejezetben összevetem a tervezés során várt és megvalósítás során kialakult eredményeket. Kitérek ez egyes alkalmazások hibáira, valamint a fontosabb eredményekre is. Emellett összehasonlítom az egyes mérések során használt eszközöket, kiemelem azok előnyeit és hátrányait, valamint megbecsülöm, milyen méretű és komplexitású alkalmazások esetén, mely eszköz használata ajánlott.

Abstract

Due to the ever increasing popularity of the web, web applications have been exposed to an increasing number of visitors. The growing number of visitors also means an increasing load, for which, in many cases, applications aren't prepared. In this case the use of more powerful hardware can be a solution for some time, but after a while we are likely to meet some kind of limit, that hinders us from further scalability. Therefore the goal of my thesis is to study methods and software, with which the performance of the web applications can be measured and improved.

In the first two chapters of my essay, I go through the background information, which are necessary for measuring and optimizing performance, and developing scalable web applications. I mention well-tried performance optimization methods, I show different software, that can make it lot easier to measure, to test web applications and to analyze the results.

In the third chapter, I plan the measurement environment and the performance optimization process of three different applications. I choose the web applications and the used softwares in order to have simple and more complex ones, and commercial and open source solutions as well. I set up the three different measurements according to three different aspects, depending on whether a user, an administrator or a developer does the optimization.

In the fourth's chapter I show the process and the important steps of the performance measurement and the results gained during the measurement in all the three aspects. I try to show each case in a way so they can be easily applied to the measurement of other web applications, and so that they are in concert with what I planned in the previous chapter.

Finally in the fifth's chapter, I compare the results expected during the planning and the results gained during the implementation. I also mention the faults of each application, and the important results as well. Furthermore I compare the tools used during each measurement, and I highlight their advantages and disadvantages, and I estimate which software is advised to use in case of applications with different size and complexity.

Tartalomjegyzék

Összefoglaló	2
Tartalomjegyzék.....	4
1 Bevezetés.....	7
2 A teljesítményoptimalizálás elméleti háttere	9
2.1 A teljesítmény definiálása.....	9
2.2 A skálázhatóság definiálása	10
2.3 Teljesítményoptimalizálás	11
2.3.1 Mit jelent a szűk keresztmetszet?	11
2.3.2 A teljesítményoptimalizálás folyamata.....	12
2.3.3 Különböző mérési módszerek ismertetése	14
2.4 Teljesítményoptimalizálási eszközök.....	16
2.4.1 A terhelésgenerálás eszközei.....	16
2.4.2 A teljesítménymérés eszközei.....	20
2.4.3 Az analízis eszközei	23
3 A teljesítményoptimalizálási mérések megtervezése	25
3.1 A teljesítményoptimalizálás általános menete.....	25
3.1.1 Tesztkörnyezet meghatározása	25
3.1.2 Teljesítménykritériumok meghatározása	25
3.1.3 Tesztesetek tervezése.....	26
3.1.4 Tesztkörnyezet beállítása	26
3.1.5 Teszt futtatása	26
3.1.6 Eredmények elemzése, tesztelés újratekintése	26
3.2 Teljesítményoptimalizálás bemutatása mintaalkalmazásokon.....	26

3.2.1	Optimalizálás felhasználói szempontból	27
3.2.2	Optimalizálás üzemeltetői szempontból	27
3.2.3	Optimalizálás fejlesztői szempontból	28
4	A teljesítményoptimalizálási mérések eredményei	30
4.1	Teljesítményoptimalizálás felhasználói szempontból	30
4.1.1	Tesztkörnyezet meghatározása	30
4.1.2	Teszteset rögzítése	31
4.1.3	Teljesítménymérés egy virtuális felhasználóval	31
4.1.4	Teljesítménymérés tíz virtuális felhasználóval	33
4.2	Teljesítményoptimalizálás üzemeltetői szempontból	34
4.2.1	Tesztkörnyezet meghatározása	34
4.2.2	Teszteset rögzítése	35
4.2.3	Egyszerveres mérés	36
4.2.4	Egyszerveres mérés VirtualPC-vel.....	38
4.2.5	Különálló web és adatbázis szerver	38
4.2.6	Két webszerver terhelés elosztással	40
4.3	Teljesítményoptimalizálás fejlesztői szempontból.....	41
4.3.1	Tesztkörnyezet meghatározása	41
4.3.2	Modul szintű teljesítménymérés Performance Tester segítségével	42
4.3.3	Forráskód szintű teljesítménymérés Performance Profilerrel	45
5	Értékelés	49
5.1	Teljesítményoptimalizálás felhasználói szempontból	49
5.2	Teljesítményoptimalizálás üzemeltetői szempontból	49
5.3	Teljesítményoptimalizálás fejlesztői szempontból.....	50

5.4	Alkalmazott szoftverek összehasonlítása	51
6	Konklúzió	53
7	Irodalomjegyzék	55

1 Bevezetés

A web fejlődésével párhuzamosan a felhasználók egyre nagyobb igényeket támasztanak a webalkalmazások iránt. A digitális könyvtárak, a távoktatás, az elektronikus áruházak mind szerepet játszanak az internet és a hálózati forgalom növekedésében. A virtuális áruházakban könyveket, autókat, számítógépeket és sok más terméket vagy szolgáltatást vásárolhatunk, emellett a legtöbb kormányhivatal lehetőséget biztosít ügyeink elektronikus intézésére. A nagyobb rendelkezésre állás és a gyorsabb kiszolgálás mellett fontos az olcsó üzemeltetés is. Gyakran találkozunk olyan portálokkal, melyek nem voltak felkészítve valós terhelésre, ilyenkor vagy nagyon lassan, vagy sehogy sem tudjuk a kívánt információt megszerezni. Ilyenkor ideig-óráig megoldást jelenthet egyre erősebb hardver használata, de egy idő után biztos beleütközünk valamilyen korlátba, ami megakadályozza a további skálázhatóságot.

Az üzemeltetők egyik legnehezebb problémája olyan informatikai technológiai infrastruktúra létrehozása, hogy az a felhasználók Quality of Service (QoS) igényeinek megfeleljen. Ez a kihívás szükségessé teszi a webalkalmazások teljesítménymérését és optimalizálását. A terhelés mérése, szűk keresztmetszetek felderítése vagy jövőbeli erőforráshiányok előrejelzése mind szükséges ahhoz, hogy meghatározzuk a legköltséghatékonyabb megoldást a növekvő terhelési igények kiszolgálására és teljesítmény problémák leküzdésére [1].

A folyamatos kapacitástervezés hiánya váratlan szolgáltatás kieséshez és teljesítmény problémákhoz vezet. Adott QoS szint fenntartása mellett nem triviális feladat további felhasználók kéréseinek kiszolgálása. A számítógépes rendszer felhasználószámának folyamatos növekedése és azonos QoS biztosítása mellett egyre több erőforrás (pl.: szerverek, kommunikációs vonalak, adattároló eszközök) felhasználását teszi szükségessé. Annak ellenére, hogy a cég anyagi forrásai lehetővé teszik a kapacitás problémák megoldására újabb hardverek és/vagy szoftverek vásárlását, megfelelő eszközök kiválasztása, kiszállítása, telepítése sok időt vesz igénybe. Ez alatt az idő alatt, az alacsony

teljesítmény miatt, a felhasználók elégedetlenek lesznek, ezzel rontva a szolgáltatásról (cégről) kialakult képet.

Egyre növekvő felhasználói igényeket csak úgy lehet kiszolgálni, ha már az alkalmazás tervezésekor figyelembe vettük a skálázhatóságot, és a fejlesztés során teljesítménymérésekkel meggyőződtek annak eredményéről. Különösen fontos lehet ez például video-on-demand alkalmazásoknál, ahol a minőségi paraméterek (késleltetés, késleltetés ingadozása, stb.) elsődleges szempontok.

A szakdolgozat célja tehát, hogy áttekintsem a teljesítménymérés és optimalizálás alapvető módszereit és eszközeit. Az elméleti áttekintés után gyakorlati szemszögből közelítem meg a témakört, és a különböző módszereket példaalkalmazásokon végzett mérések segítségével próbálom ki, melynek során többféle teljesítményoptimalizálást elősegítő eszközt tervezek megismerni.

2 A teljesítményoptimalizálás elméleti háttere

Ebben a fejezetben áttekintem azokat a módszereket és eszközöket, amik a webalkalmazások teljesítményméréséhez, optimalizálásához és skálázhatóságához szükségesek. Kitérek különböző optimalizáló módszerek megértéséhez kulcsfontosságú definíciókra és folyamatokra, valamint röviden bemutatok minden olyan eszközt, amit a webalkalmazások optimalizálása során felhasználtam. A teljesítményoptimalizáláshoz kapcsolódó tevékenységeknek ezen kívül igen komoly elméleti és matematikai módszerei vannak (pl. Markov-láncok, sorbanállási hálózatok) [2], azonban ezek megismerése és alkalmazása túlmutat jelen szakdolgozat keretein, így ezekkel dolgozatomban nem foglalkoztam.

2.1 A teljesítmény definiálása

A teljesítmény annak a jellemzője, hogy az alkalmazás adott idő és erőforrás felhasználása mellett mennyi hasznos munkát végez el. A teljesítmény többek között a válaszidővel, áteresztőképességgel és erőforrás kihasználtsággal jellemezhető [3].

Bármikor, amikor alkalmazást tervezünk, fejlesztünk, tesztelünk vagy menedzselünk, figyelembe kell vennünk a teljesítményt. Ha a szoftver nem teljesíti a vele szemben támasztott teljesítménybeli kritériumokat, az alkalmazás valószínűleg sikertelen lesz. Azonban mindaddig, amíg nem ismerjük ezeket a teljesítménybeli célokat, valószínűleg a szoftver nem is fogja elérni azokat.

A teljesítmény a fejlesztésben résztvevő különböző szereplőkre különbözőképpen hat [4]:

- Tervezőként meg kell találni a teljesítmény és skálázhatóság valamint a Quality-of-service (QoS) paraméterek között a megfelelő egyensúlyt.
- Fejlesztőként tudni kell, hogy hol kell hozzá kezdeni, hol kell folytatni és mikor kell befejezni az optimalizálást.

- Tesztelőként ellenőrizni kell, hogy az alkalmazás képes-e az előirt terhelés mellett a minőségi paramétereket teljesíteni.
- Üzemeltetőként tudni kell, mikor éri el az alkalmazás azt az állapotot, hogy már nem teljesíti a szolgáltatási szint szerződést (Service Level Agreement, SLA), valamint képesnek kell lenni növekedési tervet készíteni.

2.2 A skálázhatóság definiálása

A skálázhatóság a rendszer azon tulajdonsága, hogy a rendszer teljesítménye a rendelkezésre álló erőforrásokkal lineáris arányban változik [4]. Egyszerűbben tehát a skálázhatóság azt jelenti, hogy a rendszer teljesítménye új erőforrások felhasználásával nagymértékben növelhető.

Ha például egy skálázható rendszerben kétszeresére növeljük a processzor sebességét, akkor a kiszolgálható felhasználók száma is közel a kétszeresére növekszik. Ez nyilvánvalónak tűnik, azonban egy nem skálázható rendszerben ez nem feltétlen van így, mert ott a terhelés növelése megsokszorozhatja az erőforrás szükségleteket vagy a rendszer valami egyéb szűk keresztmetszet miatt nem tudja kihasználni a rendelkezésre álló erőforrásokat.

Egy adott alkalmazás skálázására alkalmas módszereket két kategóriába sorolhatjuk, a függőleges irányú skálázásra (*Scale vertically, scale up*) és a vízszintes irányú skálázásra (*Scale horizontally, scale out*). A függőleges skálázás során a rendszer egy kiválasztott eleméhez adunk új erőforrást, tipikusan egy számítógépet bővítünk processzorral, memóriával vagy háttértárral. A vízszintes skálázás esetén a rendszert bővítjük egy új elemmel, például számítógéppel vagy routerrel. Skálázás eszköze lehet például a terhelés elosztás (Load Balancing), melynek során kettő vagy több számítógép, hálózati kapcsolat, processzor, merevlemez vagy más erőforrás között osztjuk meg a terhelést, optimális erőforrás-kihasználtság, áteresztőképesség maximalizálása vagy válaszidő csökkentése érdekében.

A skálázhatóság hiánya általában visszavezethető valamely tervezés vagy a tesztelés során elkövetett hibára. Tipikus hibák melyek megakadályozzák a skálázhatóságot:

- A felhasznált hardver nem tökéletesen skálázható,
- Gyengén megtervezett alkalmazás architektúra,
- Feleslegesen zárolnak a rendszer elemei egy közös erőforrást,
- Egyszálú alkalmazás mely nem használja ki a rendszerben lévő több CPU-t,
- Nagy memória-igényű alkalmazás, mely 32 bites binárisokat használ, és így csak korlátozott memóriaterületet tud használni,
- Nagy memória-igényű alkalmazások nem fordítanak elegendő figyelmet a nem használt memória felszabadítására.

Amikor webalkalmazást fejlesztünk, törekedni kell arra, hogy rendszer skálázhatósága maximális legyen, viszont a skálázhatóság csak folyamatos teljesítménymérés és teljesítményoptimalizálás segítségével érhető el.

2.3 Teljesítményoptimalizálás

A teljesítményoptimalizálás egy iteratív folyamat. Miután különböző változtatásokat hajtottunk végre az alkalmazáson, minden esetben újra kell mérni és tesztelni, hogy az adott változtatás milyen hatással van a teljesítményre. Ezt addig kell folytatni, míg az alkalmazás (webalkalmazás) elérte a kitűzött célt, vagy úgy döntünk, nincs lehetőség további optimalizálásra, és a célszámot kell csökkentenünk.

2.3.1 Mit jelent a szűk keresztmetszet?

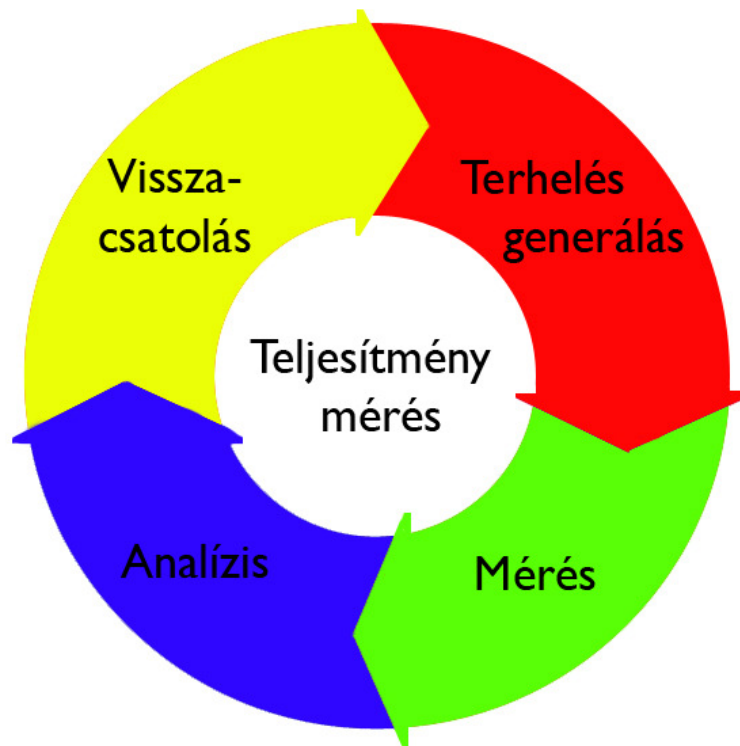
A szerverek, és a hozzájuk kapcsolódó kliensek számának növekedése mellett, a végfelhasználó által érzékelt teljesítményt általában valamely, a kliens és a szerver közti úton elhelyezkedő komponens (pl.: szerver, hálózat, link vagy router) korlátozza. Azt a komponenst, ami korlátozza a rendszer teljesítményét szűk keresztmetszetnek hívjuk. Szűk keresztmetszetek felderítése a teljesítmény analízis kulcsfontosságú lépése, mert ennek során derül ki, mely komponens az, amit fejleszteni kell a teljesítmény növelése érdekében [1].

A legtöbb esetben az alkalmazás szűk keresztmetszetét valamely hardver egység teljesítménybeli hiányossága okozza, mint például CPU, memória, merevlemez, hálózati interfész vagy más külső erőforrások, mint az adatbázis kapcsolatok száma vagy a hálózat sávszélessége.

2.3.2 A teljesítményoptimalizálás folyamata

A szoftvertervezésben az teljesítményoptimalizálás során megállapítható, hogy a rendszer egyes részei hogyan viselkednek különböző terhelés alatt. A teljesítményoptimalizálás része a szoftverfejlesztésnek, azonban különböző célokra használhatjuk fel:

- bebizonyíthatjuk, hogy a rendszer biztosítja a minőségi kritériumokat,
- összehasonlíthatunk két különböző rendszert teljesítmény szempontjából,
- megállapíthatjuk, hogy a rendszer mely része (szoftver, hardver) okozza a teljesítmény csökkenést.



1. ábra – A teljesítményoptimalizálás folyamata

A teljesítményoptimalizálás egy összetett folyamat, aminek pontos menete az alkalmazási környezettől és a mért alkalmazástól függően alakul, ám a következő lépések általában mindig megtalálhatóak bennük.

2.3.2.1 Terhelés generálás

A terhelés generálás során a mérni kívánt rendszeren, a valós élethez minél inkább hasonlító terhelést állítunk elő. A terhelés paramétereinek változtatásával tudjuk a mérést finomítani, és azt a valós felhasználók cselekedeteihez fokozatosan igazítani. A terhelést előállító program komplexitásától függően beállíthatunk fokozatosan növekvő, statikus, vagy akár véletlenszerű terhelést, így a mérési összeállításunkhoz legmegfelelőbb esetet tudjuk szimulálni.

A virtuális felhasználók számának változtatása mellett többféle forgatókönyvet is definiálhatunk különböző felhasználói műveletekhez, mint például bejelentkezés, böngészés, vásárlás, vagy megjegyzés írása. A terhelés generálás során a forgatókönyveknél az eloszlását is beállíthatjuk annak érdekében, hogy a virtuális felhasználók cselekedetei minél jobban illeszkedjenek a valós felhasználókéhoz.

2.3.2.2 A teljesítmény mérése

A mérés során a mérni kívánt hardver vagy szoftverkomponensek teljesítményét rögzítjük az eltelt idő és a terhelés függvényében. A mérés előtt össze kell válogatnunk a rögzíteni kívánt elemeket, azokat, amik fontosak lehetnek a mérésünk során. A kiválasztási folyamat általában sosem egyértelmű, az egyes mérések lefuttatása után derül ki, hogy mely komponenseket érdemes esetleg mélyebben, vagy kevésbé mélyen megvizsgálni a továbbiakban.

A legtöbb mérést csak úgy tudjuk elvégezni, hogy ha a mérni kívánt rendszeren egy program fájlba vagy adatbázisba rögzíti az elemek teljesítmény adatait. Természetesen a rögzítés is rontja a rendszer hatékonyságát, mivel erőforrásokat használ (CPU, memória, háttértárak), de a legtöbb esetben eltekinthetünk tőle, mivel a rendszer egészét vizsgálva a hatása elenyésző.

2.3.2.3 A mért adatok analízise

Az előző lépésben rögzített adatokat különböző eszközök segítségével elemezzük. Az egyes elemek teljesítményei alapján következtethetünk a szűk keresztmetszetekre vagy azok feltételezett helyére.

Az analízist nagymértékben könnyítheti a rögzített adatok grafikus ábrázolása. Sokszor a diagramra ránézve egyértelműen felfedezhető, hogy mely komponens használja túlságosan nagymértékben vagy pazarlóan az erőforrásokat.

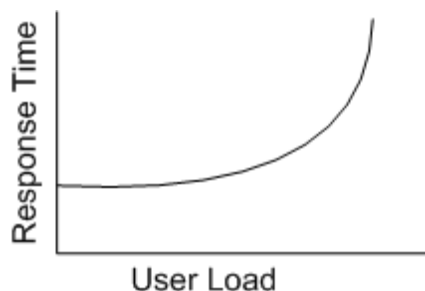
2.3.2.4 Tapasztalatok visszacsatolása

A negyedik, vagyis utolsó lépés az előző lépések során megszerzett eredmények visszacsatolása a folyamatba. Sokszor az optimalizálást úgy kezdjük el, hogy a szűk keresztmetszetet okozó erőforrás ismeretlen, és az előző tesztek eredményeit beépítve a folyamatba, fokozatosan érjük el a kívánt célt. Eredmények számíthatnak is, ha nem találtunk olyan hardverelemet, mely egyértelműen a szűk keresztmetszetet jelenti, sokszor a mérés elemzése komponensek kizárását eredményezi, így más komponenseket tudunk mélyebben vizsgálni.

2.3.3 Különböző mérési módszerek ismertetése

Teljesítményméréshez több, általánosan elfogadott módszer létezik [6], melyek segítségével több szempontból tudjuk vizsgálni a rendszerünket.

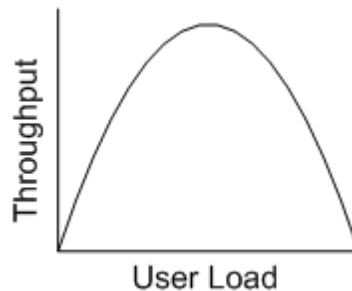
2.3.3.1 Válaszidő analízis terhelés változtatásával



2. ábra - Válaszidő és a felhasználók számának kapcsolata [6]

Amikor a válaszidőket vizsgáljuk a felhasználók számának változtatása során, meredek emelkedést kell keresnünk a válaszidőkben. Ez az emelkedő az a pont, ahol a felhasználók számának növelése a válaszidőt egyre nagyobb mértékben növeli.

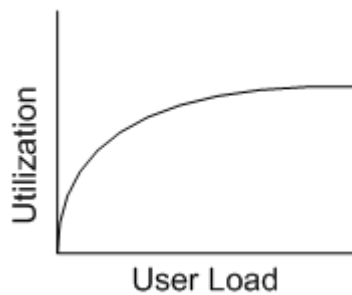
2.3.3.2 Áteresztő képesség vizsgálata terhelés változtatásával



3. ábra - Áteresztőképesség és a felhasználók számának kapcsolata [6]

Amikor az áteresztőképességet vizsgáljuk növekvő terhelés mellett, azt a pontot kell keresnünk, ahol az áteresztőképesség csökkenni kezd. Ezen a ponton elértünk egy szűk keresztmetszetet, további növelés mellett már csökkenni fog a teljesítmény.

2.3.3.3 Erőforrások terheltségének vizsgálata terhelés változtatásával



4. ábra – Az erőforrás terheltségének és a felhasználók számának kapcsolata [6]

Ebben az esetben azt vizsgáljuk, hogy mikor érik el az egyes erőforrások a 100%-os vagy közel 100%-os kihasználtságot, úgy, hogy a felhasználók számát és ezzel az összerhelést fokozatosan növeljük.

2.4 Teljesítményoptimalizálási eszközök

A mérési folyamat egyes részfeladataira különböző cégek eszközei állnak rendelkezésünkre. Minden eszköznek megvannak az előnyei és hátrányai is, találunk köztük egyaránt ingyenes eszközöket és nehezen megfizethetőket is. Mindig a feladat komplexitása függvényében döntsünk, mely programokat választjuk a méréseink során.

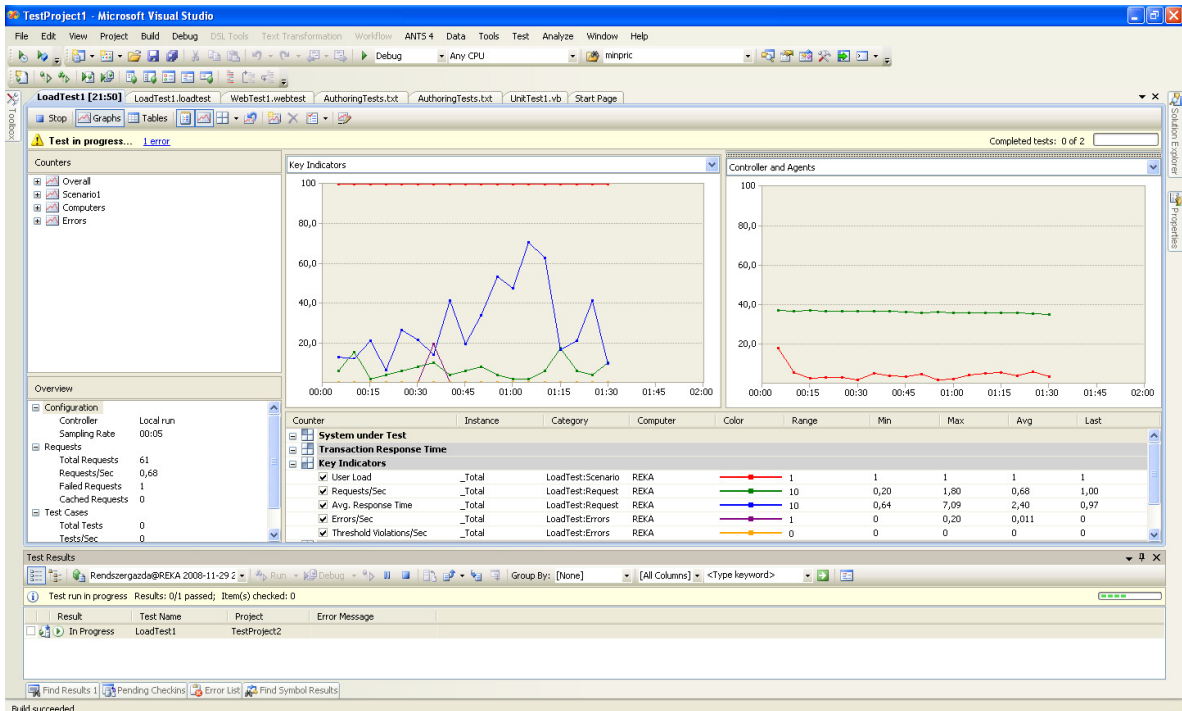
A szakdolgozat elkészítése során a következő eszközökkel ismerkedtem meg.

2.4.1 A terhelésgenerálás eszközei

A terhelésgenerálás célja, hogy a mérni kívánt rendszert virtuális felhasználók segítségével valós terhelésnek tegyük ki, és ezáltal valós környezeti feltételek mellett tudjuk az alkalmazás teljesítményét mérni és elemezni. A valós mérési eredményeket csak úgy lehetne előállítani, ha valós felhasználókkal próbálnánk terhelni a kiszolgáltót. Viszont több ezer (tíz ezer) embert bevonni a tesztelésbe nagyon költséges vagy kivitelezhetetlen feladat lehet. Ehelyett számos eszköz elérhető, melynek segítségével a felhasználók tevékenységeit szimulálni tudjuk, köztük az általam választott Visual Studio Team System, Web Test és Load Test funkciója, a Rational Performance Tester és az ingyenes FWPTT teljesítménygeneráló alkalmazás is.

2.4.1.1 Visual Studio Load Test

A *Visual Studio 2008 Team System* [7] változata biztosít funkcionalitást webalkalmazások teljesítmény vizsgálatához. Segítségével könnyedén, egy varázsló segítségével beállíthatjuk a generálandó terhelés paramétereit. A legcélravezetőbb fokozatosan növekedő teljesítmény vizsgálat, így meg tudjuk állapítani, mekkora felhasználószámnál érik el az egyes komponensek a kihasználtságuk csúcsát. Nagyobb rendszerek esetén a terhelő számítógép teljesítménye nem elegendő az alkalmazás teszteléséhez, ilyenkor több úgynevezett *Test Agent*-et hozhatunk létre és elosztottan terhelhetjük az alkalmazást, miközben az eredményeket központosítottan elemezhetjük. A Visual Studio funkcionalitása kisebb és közepesen nagy alkalmazások tesztelése esetén megfelelő, és emellett könnyen használható.



5. ábra – Visual Studio 2008 Team System -Load Test futás közben

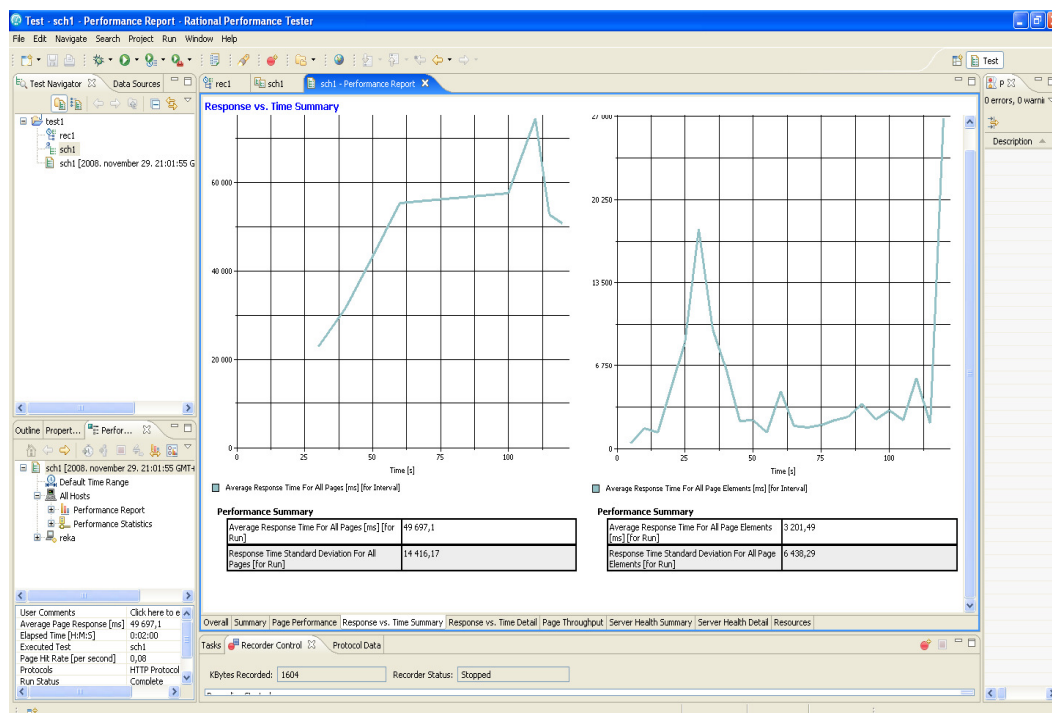
2.4.1.2 Rational Performance Tester

A *Rational Performance Tester* [8] a Rational (IBM) cég terhelésgeneráló és teljesítménymérő eszköze, mely többfelhasználós teljesítménytesztelést tesz lehetővé Windows és Linux operációs rendszeren egyaránt.

Funkciói a következők:

- A tesztek mind összesítésben, mind tételesen megtekinthetjük egy fa-struktúrájának megfelelően.
- A felhasználói viselkedést rugalmasan lehet modellezni.
- Kapacitás tervezési tesztek futtatását is támogatja az optimális hardver és infrastruktúra kiválasztása érdekében.
- Minimális hardver segítségével nagy tömegű felhasználót képes szimulálni.
- Szabványos Java tesz leírásokat használ a könnyű karbantarthatóság és kiterjesztés érdekében.
- A Tivoli eszközökkel való integráció lehetővé teszi a teljesítmény problémák gyors azonosítását.

- Opcionális kiegészítők lehetővé teszik a Siebel és SAP rendszerek tesztelését is.



6. ábra – Rational Performance Tester – egy teljesítménymérés eredményei grafikonon

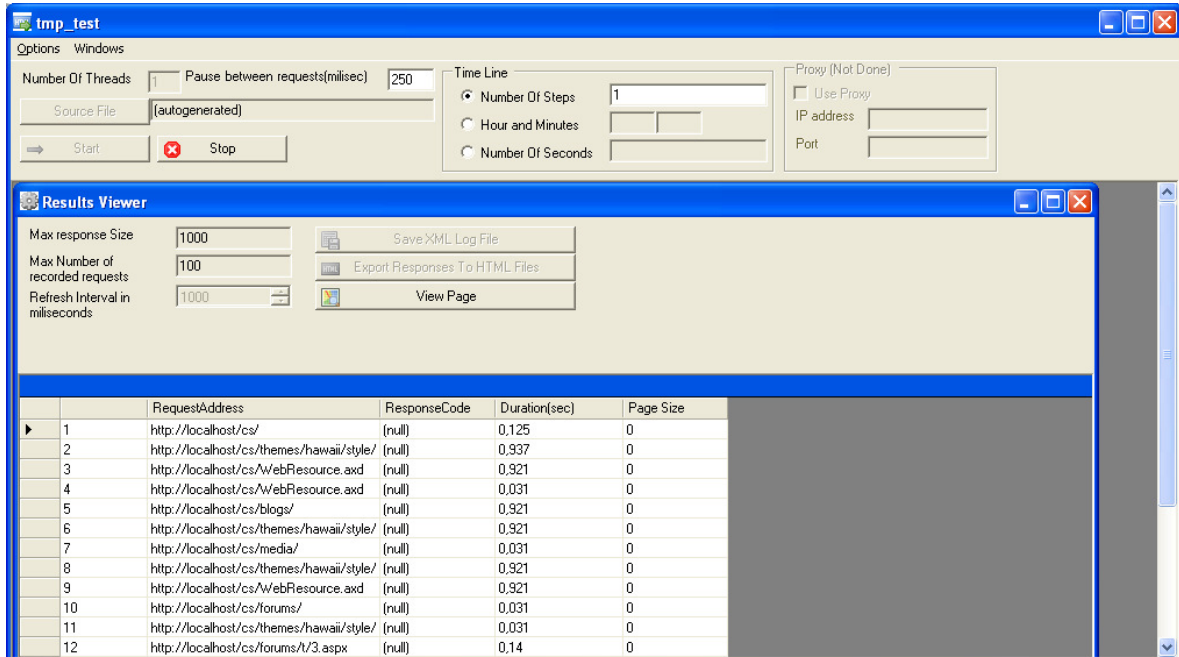
2.4.1.3 Nyílt forráskódú (OpenSource) eszköz –FWPTT Hiba! A hivatkozási forrás nem található.

Az interneten több ingyenes nyílt forráskódú tesztelő és terhelés generáló szoftver érhető el. Ezek nagy része elsősorban Apache webservereket és PHP alapú alkalmazásokat támogatja, de van köztük platform független megoldás is. Ilyen az általam megismert FWPTT program is.

Az FWPTT egy egyszerűen használható, .NET Framework felett futó program, mely webalkalmazások terheléses vizsgálatát teszi lehetővé. A normál HTTP kérések mellett támogatja az úgy nevezett AJAX (Asynchronous JavaScript and XML) kéréseket is. Bár elsősorban Internet Information Services-n futó ASP.NET webalkalmazások tesztelésére született, működik JSP, PHP és sok más programnyelvel is.

Az FWPTT a felhasználóknak lehetőséget biztosít a böngésző események rögzítésére. Ezt úgy valósítja meg, hogy létrehoz egy kliens oldali proxy-t, amin átmenő minden adatforgalmat egy XML formátumban tárol. Ezeket az XML fájlokat elmenthetjük, később

újra felhasználhatjuk, esetleg más, saját programunkba beleépíthetjük. Emellett nyílt forráskódú szoftverek előnye, hogy a program forráskódját is felhasználhatjuk saját alkalmazásinkban.



7. ábra – FWPTT program futása közben

A rögzített XML böngészési fájl teljes egészében tartalmazza a böngészőből a szerver felé indított adatokat, mint például az URL, GET és POST paramétereket. A terhelésgenerálás során ebből az XML fájlból a program egy C# osztályt generál, amit ha futtatunk, akkor a korábban rögzített forgalmat „újrajátssza”.

Mivel a program proxy szerver segítségével rögzíti az interakciót a böngészővel, bármely web böngésző alatt működik, nem vagyunk se Internet Explorer, Firefox vagy más böngészőhöz kötve. A program előnye továbbá, hogy a generált osztály, mely a terhelésgenerálást végzi, szintén nyílt forráskódú, így szabadon módosíthatjuk a saját igényünk szerint. Más, nem nyílt forráskódú szoftver segítségével elképzelhetetlen lenne például, hogy az eredményeket egy saját adatformátumba exportáljuk, azonban az FWPTT program módosításával ez is megoldható.

Alább a program által rögzített XML egy részlete látszódik:

```
<Item>
  <GSR Seri_No_="15" State_="5" In_Edit_="0" HasOrig_Vals_="0" />
  <URL>http://localhost/cs/login.aspx</URL>
  <Port>80</Port>
  <RequestType>POST</RequestType>
  <Index Null="1" />
  <QueryParams>
    <Item>
      <GSR Seri_No_="16" State_="5" In_Edit_="0" HasOrig_Vals_="0" />
      <ParamName>ReturnUrl</ParamName>
    </Item>
  </QueryParams>
  <PostParams>
    <Item>
      <GSR Seri_No_="17" State_="5" In_Edit_="0" HasOrig_Vals_="0" />
      <ParamName>__EVENTTARGET</ParamName>
      <ParamValue Null="1" />
    </Item>
  </PostParams>
</Item>
```

2.4.2 A teljesítménymérés eszközei

Bár a teljesítményadatok rögzítésére számos eszköz áll rendelkezésünkre, a legtöbb operációs rendszer saját teljesítményrögzítő és elemző szolgáltatással rendelkezik. Méréseim során a Windows operációs rendszerekben megtalálható teljesítményszámlálók adatait használtam fel.

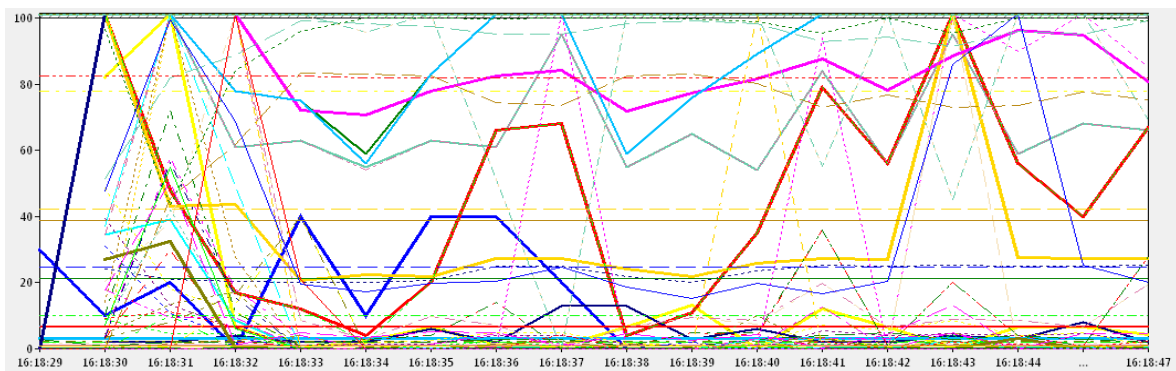
2.4.2.1 Teljesítményszámlálók

A Windows rendszerek *teljesítményszámlálói* (performance counters) a különböző rendszerkomponensek teljesítményadatainak gyűjtését végzik. A Windows számos előre gyártott teljesítményszámlálót tartalmaz, amelyek az operációs rendszerhez tartozó objektumok (hardver és szoftver) adataival foglalkoznak. Minden számláló a rendszer meghatározott funkciójához kapcsolódik, így találhatunk például a processzorra, a memóriára, a lemezegységekre, rendszerfolyamatokra és szálakra vonatkozó számlálókat is. Olyan számlálókat is használhatunk, amelyek nem az operációs rendszerhez tartoznak,

ezeket a rendszerre telepített különféle alkalmazások hozzák létre, hogy a felhasználók nyomon követhessék az adott alkalmazással kapcsolatos teljesítményadatokat.

A számlálók típusa meghatározza azt, hogy a számláló milyen számítás eredményét adja vissza, ha a számított értéket kérjük el tőle. Számos típus létezik, amelyek mindegyike az alábbi öt csoport valamelyikébe sorolható.

- Átlagszámlálók (Average counters)
- Különbségszámlálók (Difference counters)
- Pillanatnyi érték számlálók (Instantaneous counters)
- Százalékszámlálók (Percentage counters)
- Gyakoriság számlálók (Rate counters)



8. ábra- Mérés során összeválogatott teljesítményszámlálók grafikonja

Teljesítményméréskor nem szükséges az összes rendelkezésre álló teljesítményszámlálót vizsgálni, ki kell választani közülük azokat, amik fontos szerepet játszanak a mérésben. Más számlálókat kell elemezni adatbázis kiszolgáló vizsgálatok, vastag kliens alkalmazás vagy éppen webalkalmazás mérése során is.

Webalkalmazások méréséhez áttekintettem, hogy mely számlálókat érdemes vizsgálni a különböző hibák felderítésére.

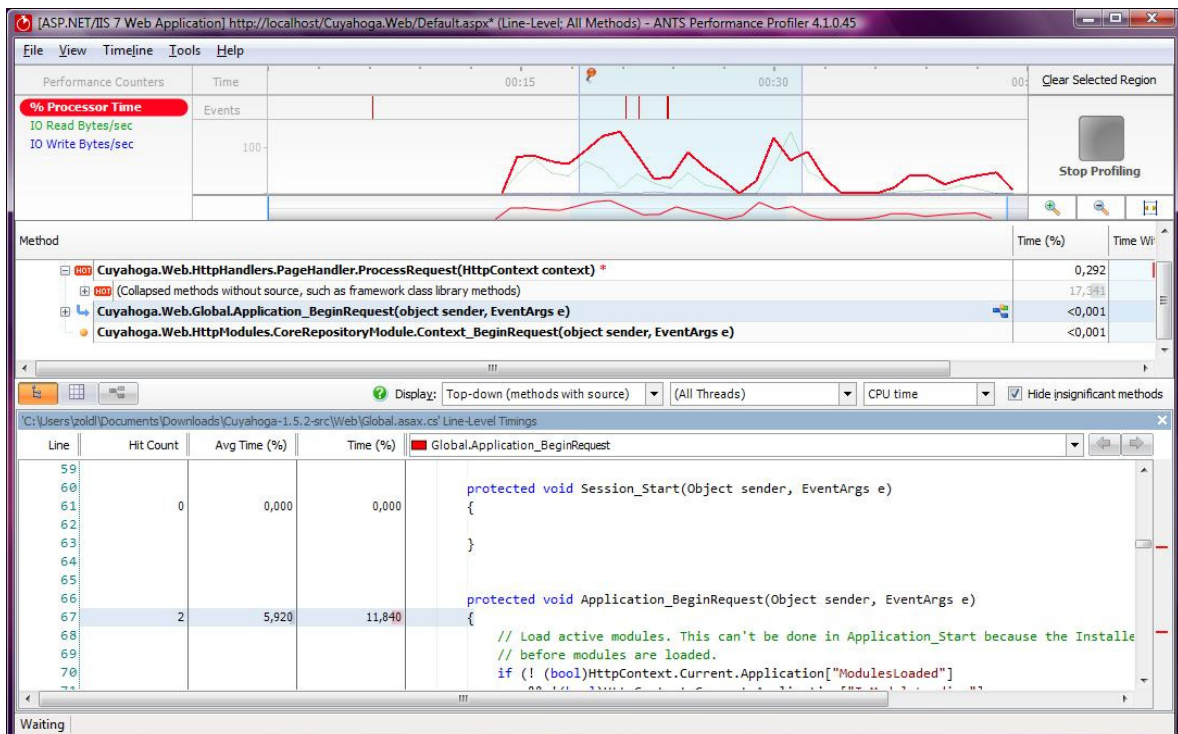
A leghasznosabb teljesítményvizsgálók webalkalmazások mérésekor

Processor(_Total)\% Idle Time	A processzor szabad erőforrásának mértéke
--------------------------------------	---

Processor(_Total)\% Processor Time	A processzor aktuális kihasználtsága
ASP,NET v2,0,50727\Request Execution Time	Web kérések végrehajtási ideje
SQLServer:Databases(_Total)\Transactions/sec	Adatbázis tranzakciók másodpercenként
SQLServer:General Statistics\Logins/sec	Adatbázis bejelentkezések száma
Memory(_Total)\Available Memory	Rendelkezésre álló memória
ASP,NET\Requests Current	Aktuális kérések száma

2.4.2.2 ANTS Performance Profiler

Az ANTS Profiler [10] .NET alapú alkalmazások mérésére és elemzésére szolgál. Az ANTS Profiler képes bármilyen .NET nyelven írt Windows Forms alkalmazást, ASP.NET webalkalmazást és Windows szolgáltatást egyaránt elemezni.



9. ábra – ANTS Performance Profiler futás közben

Az eszköz amellett, hogy könnyen kezelhető és használata könnyen megtanulható, professzionális módon nyújt segítséget az alkalmazásunk lassú részeinek felderítésére. Használatához nincs szükség szkript nyelv megtanulására, vagy használati útmutató olvasására, és az adatok elemzését már a mérés alatt elkezdhetjük, azok szinte valós időben elérhetőek.

A mért adatok egy részének kiválasztásában segít az idővonal (Time line), mely tartalmazza a rögzített teljesítményszámlálók értékét is, és azokat diagramon megjeleníti. Ahhoz, hogy későbbiekben tudjuk, hogy a rögzítés alatt milyen műveleteket végeztünk, az idővonalon címkével láthatunk el részeket.

Emellett az elemzést nagymértékben segíti a három különböző nézet: a fa, a táblázatos illetve a grafikus megjelenítés. A fa nézetben egyes függvények lefutási idejét követhetjük nyomon, és a fa fokozatos kibontásával juthatunk el alacsonyabb szintű metódusokhoz. A táblázatos nézet tartalmazza az összes mért metódus vagy függvény lefutási idejét, számát és forrásfájlját. Futási idő szerint sorba rendezve rögtön látjuk, mely metódusok voltak a leglassabbak. A grafikus nézet segítségével vizuálisan tudjuk egyes metódusok futási idejét megtekinteni.

2.4.3 Az analízis eszközei

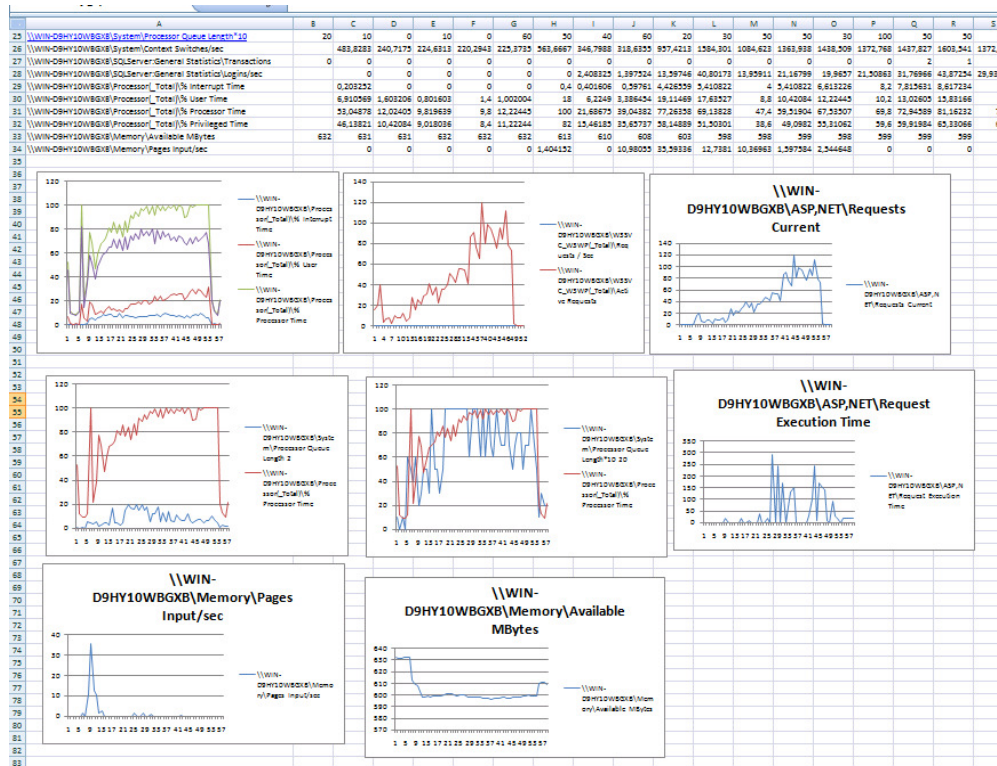
Az analízis során a teljesítmény adatokat elemezzük többféle szempont szerint. Megpróbálunk minél több értékes eredményt kigyűjteni, és ebben sokat segítenek a különböző eszközök.

2.4.3.1 Visual Studio

A Visual Studio [7] is nyújt beépített funkcionalitást adatok elemzésére. A mérés befejezése után egy összefoglaló lapon tekinthetjük át a fontosabb eredményeket, pl. a sikeres tesztek számát, vagy a hibák okait és a válaszidőket. A mérés közben rögzített adatokat egy SQL adatbázisba menti, így a mérés befejezése után bármely számlálót újra elővehetünk és elemezhetünk. A program képes grafikonon is ábrázolni a számlálók adatait, ezzel megkönnyítve az analízist.

2.4.3.2 Microsoft Office – Excel

Ha a teljesítménytesztelő alkalmazások előre definiált táblázatainál és grafikonjainál bővebb információra van szükségünk, egy táblázatkezelő szoftver és a mért adatok felhasználásával egyszerűen definiálhatunk saját diagramokat. Saját méréseim és azok elemzése során is az Excel-t részesítettem előnyben. [13]



10. ábra - Adatok elemzése Excel segítségével

A teljesítményszámlálók által rögzített adatokat CSV (Comma Separated Value) formátumba exportáltam, melyet az Excel már meg tudott nyitni. Az általam fontosabbnak tartott számlálókhoz grafikonokat készítettem, és így a száraz számok helyett a grafikonokat elemeztem. Fontos volt számomra, hogy szabadon kombinálhattam a grafikonokon megjelenítendő számlálókat, így például közös grafikonon ábrázolhattam a várakozó kérések számát és a válaszidőt, vagy a processzor kihasználtságot és a kérések feldolgozási idejét.

3 A teljesítményoptimalizálási mérések megtervezése

A webalkalmazások teljesítményoptimalizálást több különböző szempontból lehet megközelíteni, attól függően, hogy az alkalmazás mely részét és azt milyen szinten tervezzük optimalizálni.

3.1 A teljesítményoptimalizálás általános menete

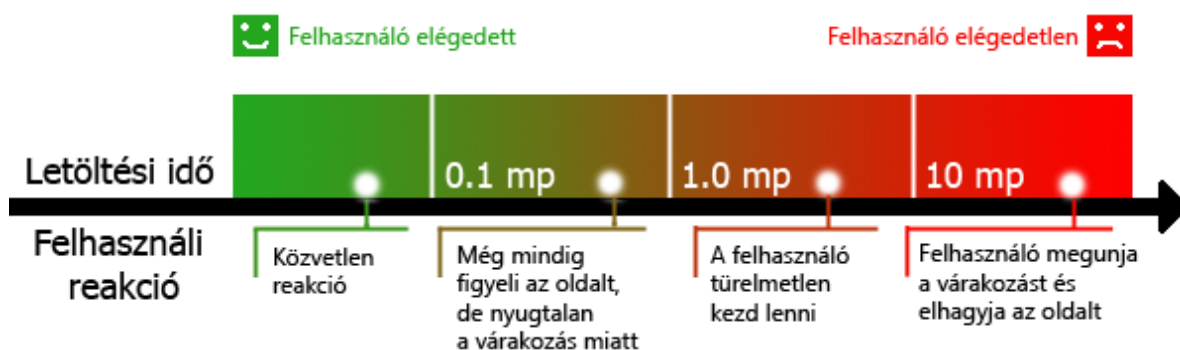
Bár az egyes alkalmazásokat más-más szempontból, más-más eszközökkel és erőforrások segítségével vizsgálják, az optimalizálás folyamata mindig közös sémára épül [3].

3.1.1 Tesztkörnyezet meghatározása

Első lépésben meghatározzuk a fizikai tesztkörnyezetet, a tesztelők rendelkezésére álló eszközöket és erőforrásokat. A fizikai környezet alatt a hardvert, szoftvert és hálózati beállításokat értem. Ha átfogó képet szerzünk a tesztkörnyezetről, az a későbbiekben megkönnyíti és hatékonyabbá teszi a tesztelést, és segít, hogy pontos célokat definiáljunk már a projekt kezdetekor. Néhány esetben ehhez a folyamathoz újra és újra vissza kell térnünk egy projekt életciklusa során.

3.1.2 Teljesítménykritériumok meghatározása

A teljesítménykritériumok alatt elsősorban a válaszidőt, áteresztőképességet, az erőforrás használati célokat értjük. Általánosságban véve, a válaszidő a felhasználó, az áteresztőképesség a tulajdonos, míg az erőforrás kihasználás a rendszer érdeke.



11. ábra - Felhasználó reakciója válaszidő függvényében

3.1.3 Tesztesetek tervezése

Miután meghatároztuk a teljesítményi kritériumokat, a tesztesetek tervezése következik, melynek során meghatározzuk a kulcsfontosságú forgatókönyveket (*scenario*), tipikus felhasználói tevékenységeket, teszt adatokat, és a mérés folyamán figyelni kívánt metrikákat.

3.1.4 Tesztkörnyezet beállítása

A következő lépésként előkészítjük, majd beállítjuk az előzőekben meghatározott tesztkörnyezetet, a hozzá tartozó eszközöket és az erőforrásokat. A tesztkörnyezet beállítása során törekedni kell arra, hogy az egyes beállítások a valós környezethez leginkább illeszkedjenek.

3.1.5 Teszt futtatása

Ezt követően tesztelő programok segítségével lefuttatjuk a teszteseteket, és erőforrás kihasználtságot folyamatosan rögzítjük. A legtöbb eszköz már a futás közben lehetőséget biztosít a mért adatok elemzésére és rendszer állapotának követésére. Hiba vagy nem várt működés esetén a tesztet le kell állítani, és újrafuttatni.

3.1.6 Eredmények elemzése, tesztelés újratekintése

Az optimalizálás utolsó szakaszában a különböző tesztesetek eredményeit egy közös modellbe egyesítjük. Ha valamely modul mérése nem sikerült, vagy az eredmények nem egyértelműek, finomhangolva a paramétereket meg kell ismételni a mérést. Az egyes mérések során szerzett információkat be kell építeni a folyamatba, ha az eredmények megegyeznek az előre definiáltakkal, akkor befejezettnek tekinthetjük a mérést. Abban az esetben, ha még nem értük el a kívánt cél, a rendszer és tesztkörnyezet módosításával további mérések végezhetünk.

3.2 Teljesítményoptimalizálás bemutatása mintaalkalmazásokon

Az előzőekben bemutatattam a teljesítményoptimalizálás és mérés általános menetét. A továbbiakban az elméletet gyakorlatban is kipróbálom, ehhez különböző

mintaalkalmazásokon valós teljesítményméréseket fogok végezni. A méréseket a következő szempontok és eszközök felhasználásával fogom végrehajtani.

3.2.1 Optimalizálás felhasználói szempontból

Első lépésben a felhasználó szintű optimalizálás lehetőségeit nézem át. Sok esetben a webalkalmazás üzemeltetőjének nem állnak rendelkezésére az architektúráis optimalizálás és skálázás eszközei, illetve nincs lehetősége forráskód alapú analízisre (vagy azért mert nem rendelkezik a forráskóddal, vagy csak nincs meg a megfelelő tudása hozzá).

Emellett kis költségvetésű projektek esetén nincs lehetőség neves cégek tesztelő, illetve optimalizáló szoftvereinek megvásárlására, mivel azok licenc díjai meghaladhatják a teljes alkalmazásfejlesztés költségvetését. Azonban ezekben az esetekben sem hanyagolható el alapszintű tesztelés és terheléses vizsgálat.

Az előzőeknek megfelelően egy olyan tesztkörnyezetet tervezek kialakítani, ami egyetlen PC-t tesz szükségessé, illetve a teszteléshez egy ingyenes, nyílt forráskódú szoftvert fogok használni. Ebben az esetben azt szeretném kideríteni, mi az, ami megvalósítható ingyenes eszközzel, és mi az a pont, amikor a projekt egy fizetős szoftver vásárlását teszi szükségessé. Feltételezésem szerint az ingyenes szoftver segítségével is el lehet végezni az alapfeladatokat, legfeljebb kicsit körülményesebben, vagy nem olyan hatékonyan.

3.2.2 Optimalizálás üzemeltetői szempontból

Üzemeltetői optimalizálásnak nevezhetjük azokat a megoldásokat, amikor a szoftver architektúra, a forráskód vagy az algoritmusok módosítása helyett megpróbáljuk a rendszert kiszolgáló infrastruktúra módosításával növelni a hatékonyságot (tehát például megváltoztatjuk a hálózati topológiát vagy további kiszolgálókat állítunk üzembe).

A web szolgáltatásokat megvalósító rendszerek általában különböző egymásra épülő szoftver komponensekből épülnek fel. A kéréseket egy webszerver szolgálja ki, amely az adatokat általában relációs adatbázisokból, vagy más rendszer specifikus adatbázisból kapja. A rendszert kiszolgáló szoftverek (szoftverkomponensek) csak kevés esetben futnak

egy közös szerver gépen, általában több különálló, egymás között nagysebességű hálózaton kommunikáló gépet használnak.

Kezdetben a teljesítménymérést egyszerveres környezetben tervezem mérni, ahol a webservert és az adatbázis szerver azonos gépen található. A következő lépésekben fokozatosan finomítom a teszt elrendezést, és azt vizsgálom, hogy mi a szűk keresztmetszet az egyes fokozatokban. Például, különválasztom a két fő funkcionalitást (adatbázis és web kiszolgálás) két külön szerverre, majd hálózati terheléelosztás segítségével többszörözöm a webszerveret.

A teljesítménymérést a Visual Studio Team Suite változatában elérhető tesztelési eszközök segítségével tervezem elvégezni. Az eszköz képes felhasználói tevékenységek rögzítésére, majd ezek alapján terhelési profilok generálására. A terheléses tesztek során kielemezem a folyamatosan növekvő terhelés esetén a válaszidőket, ez segít az infrastruktúra szűk keresztmetszetét megtalálni. A mért adatok alapján már meg lehet adni egy adott infrastruktúra és minőségi paraméterek mellett a kiszolgálható felhasználók számát.

A kismemelt mintaalkalmazás a Microsoft által publikált PetShop 4.0 [15]. Az alkalmazáshoz szükséges infrastruktúra komponenseket az Internet Information Services webservert és a Microsoft SQL server 2005 adatbázis-kiszolgáló biztosítja.

3.2.3 Optimalizálás fejlesztői szempontból

A webalkalmazásokat fejlesztői szempontból is mérhetjük. Ilyenkor rendelkezésünkre áll az alkalmazás forráskódja, így annak elemzésével próbáljuk a teljesítmény csökkenést okozó programrészeket azonosítani és optimalizálni. Ilyen szintű optimalizáláshoz szükség van az alkalmazás forráskódjára, felépítésének, illetve a program nyelvének ismeretére. Legtöbb esetben ez nem okoz problémát, mert forráskód szintű optimalizálást általában a fejlesztők a saját kódjukon végeznek, nem pedig idegen alkalmazásokon. Abban az esetben, ha nem saját alkalmazást vizsgálunk, először a működését kell megérteni, és csak utána lehet hozzákezdeni a mérésekhez.

A mérés során a *Cuyahoga* [16] nyílt forráskódú portál keretrendszert tervezem mérni. A Cuyahoga egy modulokból felépülő tartalomkezelő rendszer. A portált C# nyelven írták, de mivel teljes mértékben támogatja a Mono-t, UNIX alapú operációs rendszer alatt is üzemeltethető. A keretrendszer Microsoft SQL Server, PostgreSQL, és MySQL adatbázis kiszolgálókkal egyaránt működik, adatelérési rétegben NHibernate-t¹ használ.

Első lépésben a Rational Performance Tester 7.0 béta segítségével terhelem a portált, és megvizsgálom, hogy a keretrendszer és az egyes modulok hogyan viselkednek terhelés alatt. Véleményem szerint lesz olyan modul mely több erőforrást fog használni a többinél, illetve működése kevésbé lesz optimális.

Ezek után a modulok teljesítmény adatait felhasználva ANTS Performance Profiler segítségével forráskód szinten is megvizsgálom a modulokat, hogy mely kódrészlet okozza a teljesítmény csökkenést.

A vizsgált modulok:

- Bejelentkezés
- Regisztráció
- Blog (Cikkek)
- Galéria
- Fórum
- Letöltések
- Statikus oldal

Feltételezésem szerint, mivel a portál NHibernate-t használ az adatok elérésére, és minden lekérdezés ad-hoc jelenik meg a kiszolgálónál (nem használ tárolt eljárásokat), az adatelérés több időt fog elvenni a megszokottnál, mert így az adatbázis kiszolgáló nem tud előre végrehajtási tervet készíteni. A galéria és a letöltések modul a fájlokat nem az adatbázisban tárolják, hanem fájl szinten a webalkalmazás könyvtár hierarchiájában, így nagy fájlok esetén sem lesz teljesítménybeli probléma.

¹Az NHibernate egy nagy teljesítményű objektumrelációs perzisztencia és adatbázis lekérdező (query) szolgáltatás. A NHibernate lehetővé teszi az objektum-orientált elvet követő perzisztens osztályok létrehozását, beleértve az asszociációt, öröklődést, polimorfizmust, kompozíciót, kollekción.

4 A teljesítményoptimalizálási mérések eredményei

Az előző fejezetben a teljesítményoptimalizálást három különböző szempontból közelítettem meg. A következőkben mindhárom szempont esetén bemutatom a mérések folyamatát, eredményeit és mérések során felhasznált eszközöket. Az egyes eseteket megpróbálom úgy bemutatni, hogy az más webalkalmazás mérésére is könnyen alkalmazható legyen, és a tervezés során leírtaktól minél kevésbé térjen el.

4.1 Teljesítményoptimalizálás felhasználói szempontból

Egyszerű webalkalmazás készítésekor sokszor nincs szükség komoly bonyolultságú és komplexitású (ennek következtében magas költségű) teljesítménytesztelő alkalmazások használatára. Mivel a tesztelést még nagyon egyszerű alkalmazások esetében sem mellőzhetjük, érdemes nyílt forráskódú, ingyenesen elérhető termékeket kipróbálni. Bár ezek funkcionalitása általában nem összemérhető a multinacionális cégek termékeivel, egyszerűbb esetekben mégis elegendők. Az előzőeknek megfelelően, a felhasználó szintű teljesítményoptimalizáláshoz az FWPTT nyílt forráskódú teljesítménytesztelő alkalmazást használtam.

4.1.1 Tesztkörnyezet meghatározása

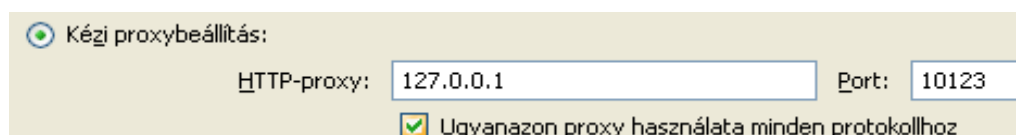
A tesztkörnyezetet úgy alakítottam ki, hogy a tesztelő gép és a tesztelendő webszerver egyazon gépen helyezkedett el. Ennek következtében a terhelés generálás csökkentette a processzor teljesítményét, és így a kapott eredmények nem feleltek meg 100%-ban a valóságnak. De ebben az esetben, ahol a cél az, hogy minél költséghatékonyabban eredményt felmutassunk, eltekinthetünk ettől a tényezőtől.

A kiszolgáló oldalon is úgy választottam meg a szoftvereket, hogy egy alacsony költségvetésű projektet jellemezzen. Ennek megfelelően a webszerver egy Microsoft Internet Information Services 5.0, ami egy Windows XP operációs rendszer alatt fut. A tesztelés során a Telligent cég nyílt forráskódú Community Server portálrendszerét

mértem. A portál ASP.NET alapokra épül és az SQL Server 2005 ingyenes Express Edition változata szolgálja ki az adatbázis kéréseket.

4.1.2 Teszteset rögzítése

A FWPTT a rögzítéshez egy kliens oldali proxy-t hoz létre, és minden, a proxy-n átmenő adatforgalmat rögzít. Ehhez a böngészőben be kell állítani, hogy minden http protokollú forgalmat a tesztelő program által létrehozott proxy-hoz irányítson. Ezen egyszerű ötlet eredményeként bármely, általunk kedvelt böngésző programot használhatjuk a teszteset rögzítésére.

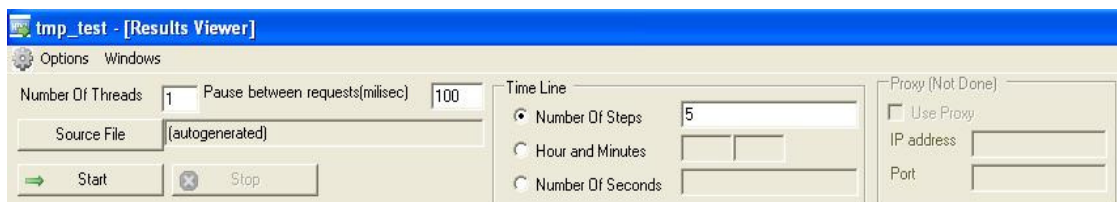


12. ábra – Proxy beállítása böngészőben

Az általunk rögzített teszteseteket XML formátumba exportálhatjuk, így később újra felhasználhatjuk, vagy más programmal elemezhetjük.

4.1.3 Teljesítménymérés egy virtuális felhasználóval

A rögzített tesztesetünket különböző terhelési beállítások mellett futtathatjuk. A mérés hosszát megadhatjuk időben, illetve futtatási számban. Az alábbi beállítás esetén a rögzített tesztesetet egy szálon egymás után 10-szer futtatja le, a kérések között 100 milliszekundum szünetet tartva. A program egyszerűsége a beállítások lapon is látszik. Nincs mód dinamikusan növekvő virtuális felhasználói szám beállítására, illetve különböző tesztesetek párhuzamos futtatására sem.



13. ábra – Egy felhasználós teszteset beállításai

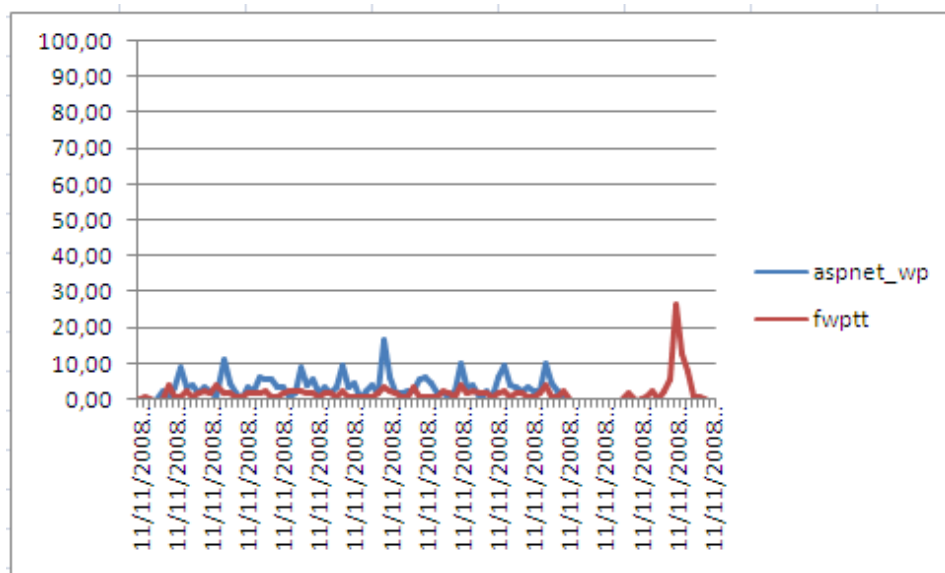
A teszteset futtatása során az eredményeket valós időben követhetjük, és hiba esetén bármikor leállíthatjuk.

	RequestAddress	Response	Duration(sec)	Page Size
89	http://localhost/cs/themes/hawaii/style/DynamicStyle.aspx	200	0	6576
90	http://localhost/cs/forums/	200	0,031	15576
91	http://localhost/cs/themes/hawaii/style/DynamicStyle.aspx	200	0	6576
92	http://localhost/cs/	200	0,046	17144
93	http://localhost/cs/themes/hawaii/style/DynamicStyle.aspx	200	0	6576
94	http://localhost/cs/logout.aspx	200	0	6341

14. ábra – FWPTT terhelésgenerálás közben

Mint a pillanatképen is látszik, a lap méretén kívül a válaszidő az, amit a program kimenetként generál. Bár a teljesítménymérés esetén ez a legfontosabb tényező, a válaszidő okára semmilyen más adatból nem következtethetünk. Nincs lehetőségünk a programban teljesítményszámlálók rögzítésére és statisztikák, diagramok, illetve jelentések készítésére sem.

Mivel a program nem nyújt lehetőséget teljesítmény adatok rögzítésére, ezért a Windows felügyeleti központ segítségével rögzítettem a CPU terheltségét. Mivel jelen esetben a terhelő és a kiszolgáló alkalmazás egy azon gépen helyezkedik el, fontos rögzíteni, hogy a processzor kihasználtsága, hogy oszlik meg az egyes folyamatok között, jelen esetben az *aspnet_wp.exe* (Internet Information Services) és a terhelő alkalmazás, az *fwptt.exe* között.



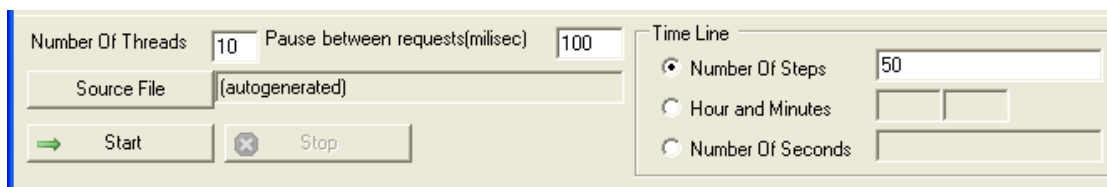
15. ábra – Processzor terheltség egy felhasználás esetén

Az előző ábra a processzor terheltséget mutatja a teszt futtatása alatt. Mint látszik a kihasználtság egy felhasználó esetén nem számottevő, a terheltség alig emelkedett 10% fölé.

A program a teszt futtatása közben folyamatosan visszajelzi az egyes kérések várakozási idejét. A teszt összes http kérésének válaszidejét átlagolva megkapjuk, hogy átlagosan mennyi idő alatt hajt végre egy kérést a webservert. Egy virtuális felhasználó esetén ez 0,02348 másodperc volt.

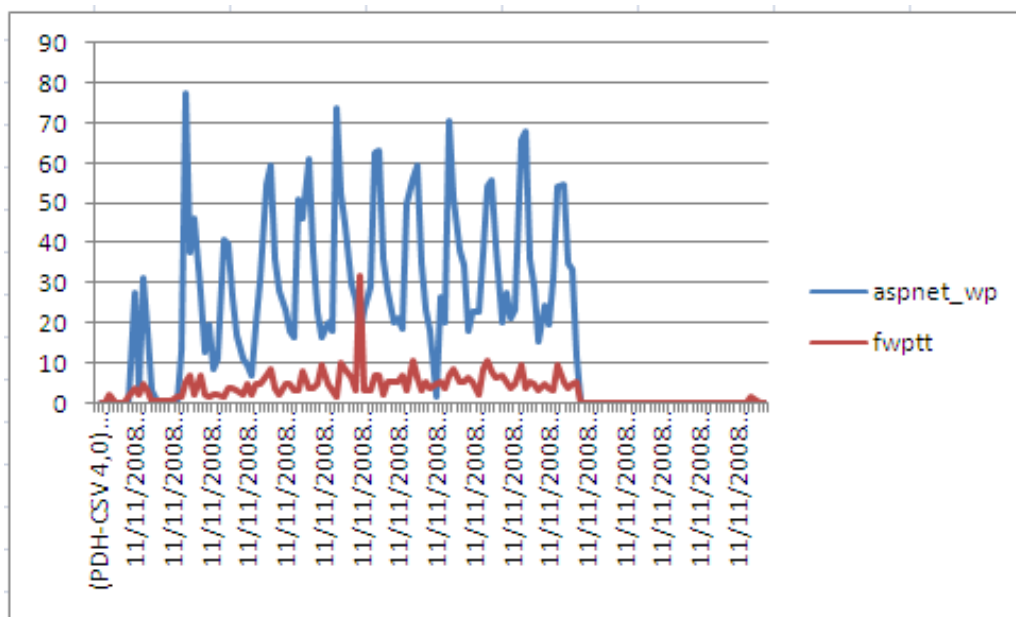
4.1.4 Teljesítménymérés tíz virtuális felhasználóval

Mivel az előző mérés során látszott, hogy nem okozott gondot egy felhasználó kiszolgálása, ezért a párhuzamos felhasználó szám növelése mellett döntöttem. A program lehetőséget biztosít a párhuzamos tesztelésre, ehhez a futtatandó szálak számát kell megnövelni. Mivel a teszt tíz párhuzamos szál mellett túl gyorsan lefutna, ezért a tesztek számát is megtízszereztem, aminek már elegendőnek kell lennie ahhoz, hogy adatokat elemezni tudjam.



16. ábra – Tíz virtuális felhasználós tesztet beállításai

Ebben a tesztetben is a két fontos folyamat (aspnet_wp, és fwptt) processzorhasználatát monitoroztam és jelenítettem meg grafikusán Excel segítségével.



17. ábra – Processzor terheltség 10 felhasználós esetben

A képen látható, hogy az elvártak szerint a terheltség megtöbbszöröződött az előző esethez képest. A válaszidők átlagát tekintve, ebben az esetben 0,06 másodpercet mértem, ami körülbelül háromszorosa az egy felhasználós esetnek, mialatt a párhuzamos felhasználók száma megtízszereződött.

4.2 Teljesítményoptimalizálás üzemeltetői szempontból

Üzemeltetői optimalizálás során a kiszolgáló infrastruktúra módosításával, javításával próbálom növelni a teljesítményt a mérések során.

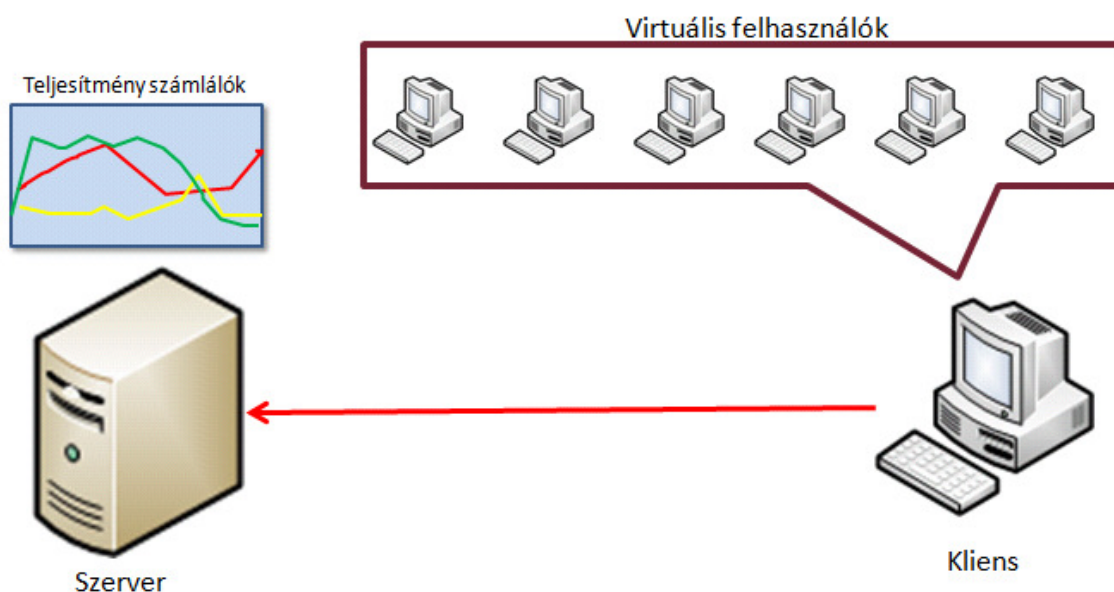
4.2.1 Tesztkörnyezet meghatározása

Az általam összeállított mérési elrendezés egy kiszolgáló gépet és egy tesztelő gépet tartalmazott. A kiszolgáló gépen Windows Server 2008 operációs rendszeren Microsoft SQL Server 2008 és Internet Information Services 7.0 szolgálta ki a Microsoft Pethop 4.0 webáruház mintaalkalmazást. A tesztelő gépen Windows Vista és Visual Studio Team System 2008 futott.

A kiszolgáló gép	Tesztelő gép
<ul style="list-style-type: none"> • Dual Xeon 3.0Ghz, 4 GB memória, Gigabit Ethernet • Windows Server 2008 • Internet Information Services 7.0 • Microsoft SQL Server 2005 • Microsoft PetShop 4.0 	<ul style="list-style-type: none"> • Core 2 Duo 2.2 Ghz, 2 GB memória, 100 Mbit Ethernet • Windows Vista • Microsoft Visual Studio 2008 Team Suit <ul style="list-style-type: none"> – Web Test – Load Test

4.2.2 Teszteset rögzítése

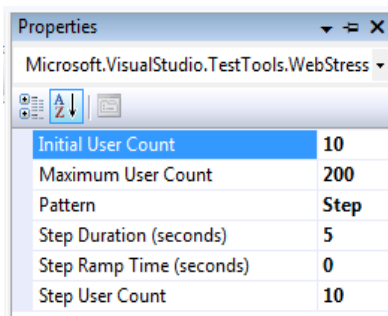
A Visual Studio Web test funkciója segítségével képesek vagyunk rögzíteni egyes felhasználók tevékenységeit, mint például bejelentkezés, keresés vagy vásárlás.



18. ábra- Mérési folyamat szemléltetése

Miután a rögzítettünk egy vagy akár több tesztesetet, a *Load test* funkció segítségével beállíthatjuk, hogy egy előzőleg rögzített esetet mekkora felhasználó számmal, milyen hosszan szeretnénk futtatni. Lehetőségünk van egyszerre több eset párhuzamos futtatására is, így valósághoz közelebb szimulációt készíthetünk, mert a valós életben a különböző felhasználók más és más célból nézik meg az adott webportált (web oldalt). A konstans terhelés mellett beállíthatjuk, hogy a felhasználók száma milyen sebességgel és

időközönként változzon, így akár lineárisan növekvő felhasználó számú teszteseteket állíthatunk össze.



Microsoft.VisualStudio.TestTools.WebStress	
Initial User Count	10
Maximum User Count	200
Pattern	Step
Step Duration (seconds)	5
Step Ramp Time (seconds)	0
Step User Count	10

19. ábra- A terhelés beállításai a Visal Studioban

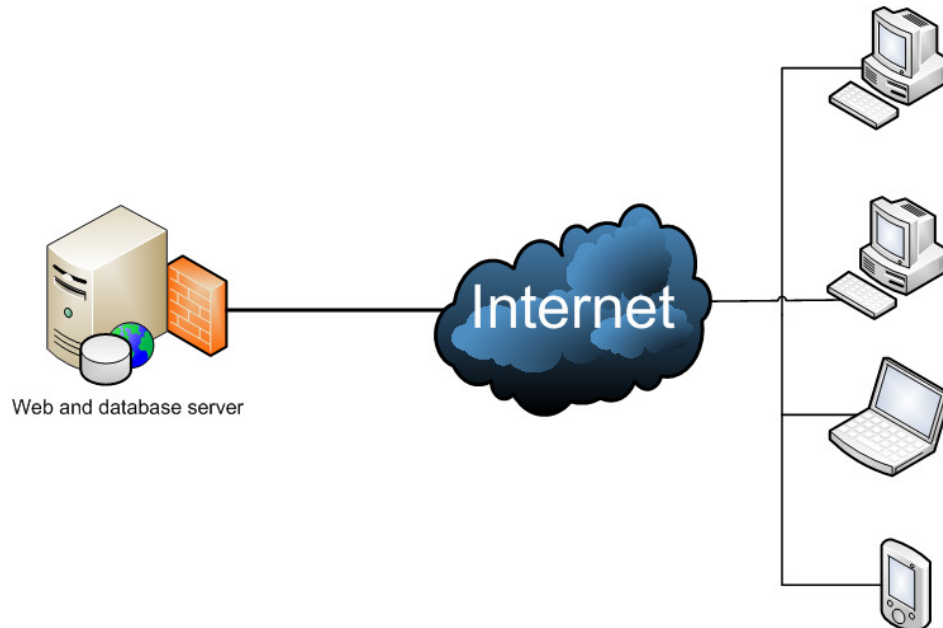
A tesztelés elindítása előtt ki kell választanunk azokat a teljesítményszámlálót, amiket rögzíteni szeretnénk a mérés során. Az összes lépés közül talán ez igényli a legtöbb időt, mivel nincs állandó, mindig érvényes séma, ami megadja, mely számlálót kell elemeznünk. Többnyire inkább ajánlások vannak, hogy webalkalmazás, vastag kliens alkalmazás vagy akár adatbázis kiszolgálás teljesítmény vizsgálta során mely számlálók lehetnek izgalmasak.

4.2.3 Egyszerveres mérés

Egyszerveres mérés esetén a webkiszolgálót (Internet Information Services) és az adatbázis kiszolgálót (SQL Server 2005) ugyanarra a gépre telepítettem fel. A szerver géphez a tesztelő gépen futó virtuális felhasználók csatlakoznak interneten (helyi hálózaton) keresztül. Ebben a konfigurációban egyetlen gép végzi az összes feladatot, így az egyes funkciók teljesítmény csökkentése hatással lehet a többi szolgáltatásra is, vagy akár teljesen megállíthatja a működést.

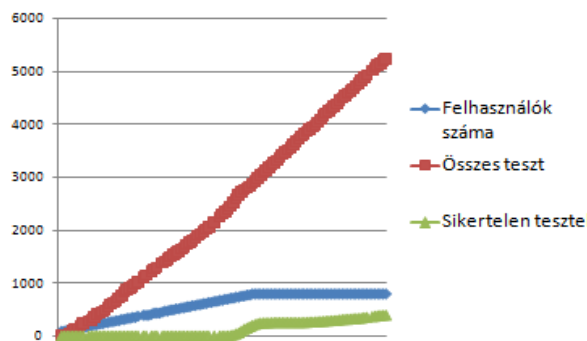
A teszt elindítása után a felhasználók száma elkezdett növekedni (21. ábra kék vonal), de egy bizonyos pont után (800 virtuális felhasználó) a sikertelen tesztek száma (21. ábra zöld vonal) is elkezdett emelkedni. A második képen látható processzor terheltségi diagramot elemezve azt vettem észre, hogy a kiszolgáló gép mindössze körülbelül 20%-ban használta a processzort (22. ábra), elenyésző memória kihasználtság mellett. Több végigfuttatott teszt után derült ki, hogy a szűk keresztmetszetet a tesztelő gép hálózati csatlóója

jelentette. Míg a szerver gép gigabites Ethernet kártyával volt felszerelve, a tesztelő gépen csak 100 megabites állt rendelkezésre, nagy mennyiségű virtuális felhasználó szám esetén nem volt képes több adatot küldeni és fogadni a hálózati csatlólón.

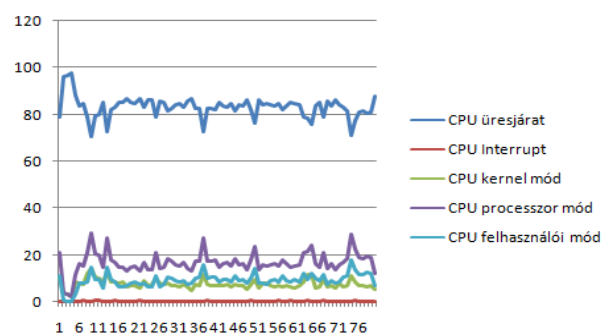


20. ábra - Egyszerveres mérési elrendezés

A hálózati eszközt gigabites kártyára kicserélve, ezzel megszüntetve a szűk keresztmetszetet, tovább folytattam a tesztelést. Hasonlóan az előzőekhez, a kiszolgáló gépet nem tudtam 20-24%-nál jobban leterhelni, pedig egy adott pont után a sikertelen tesztek száma ismét növekedett. Ekkor vettem észre, hogy a tesztelő gép CPU kihasználtsága mindkettő magon 100% közeli, tehát a kliens gép gyenge volt ilyen nagy mennyiségű felhasználó virtualizálására.



21. ábra - Felhasználók száma és tesztek eredménye

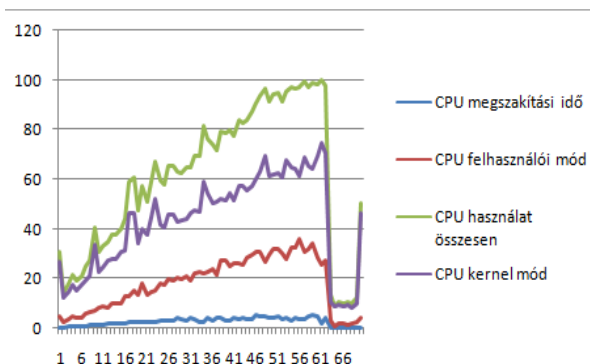


22. ábra - Kiszolgáló processzor terheltsége

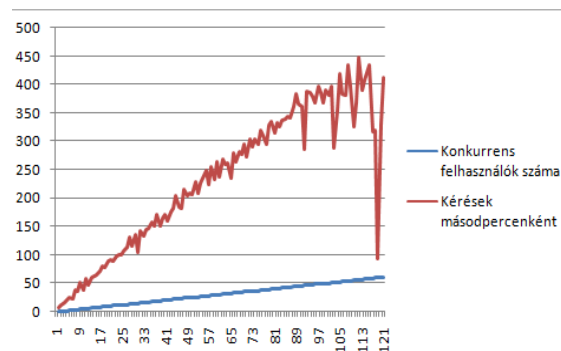
4.2.4 Egyszerveres mérés VirtualPC-vel

A gépek paraméterei adottak voltak – a kliens gép teljesítményét nem tudtam megnövelni – a kiszolgáló gép teljesítményét kellett csökkentenem. A kiszolgáló gépre feltelepítettem a Microsoft Virtual Server 2005-t, és a szolgáltatásokat (web, adatbázis kiszolgálás) egy virtuális számítógépre telepítettem, és konfiguráltam be. Mivel a virtuális számítógépek csak egy mag kapacitását tudják kihasználni, nagymértékben tudtam csökkenteni a szerver gép teljesítményt.

Az előző tesztet megismételve látszik, hogy a felhasználók számának növekedésével párhuzamosan a processzor terheltsége is növekszik, és fokozatosan eléri a 100%-t. A grafikonon látszik, hogy körülbelül 80 virtuális felhasználót tudunk e konfiguráció mellett kiszolgálni, és a szűk keresztmetszetet a processzor jelenti. Még 100%-os processzor kihasználtság mellett is elenyésző volt a merevlemez, a hálózat és processzor kihasználtsága.



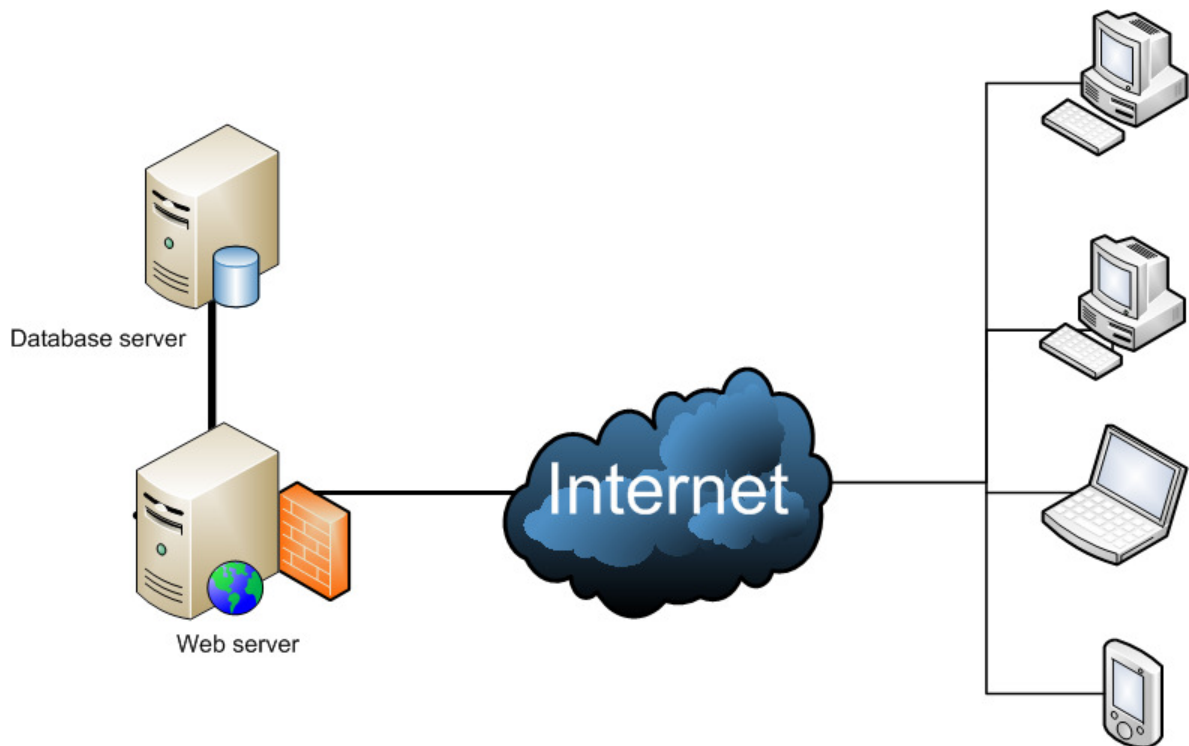
23. ábra - Kiszolgáló processzor terheltsége



24. ábra - Felhasználók száma és kérések másodpercenként

4.2.5 Különálló web és adatbázis szerver

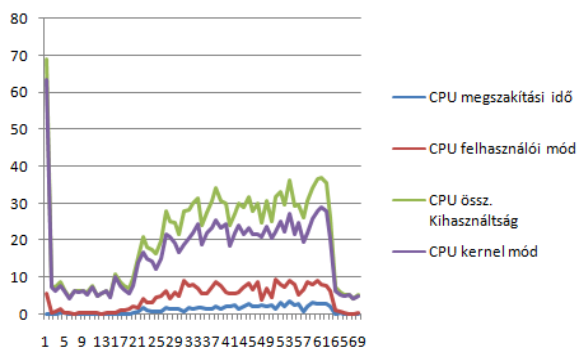
Látva, hogy a processzor jelenti a szűk keresztmetszetet, a két fő funkcionalitást (web kiszolgálás és adatbázis kiszolgálás) különválasztottam két különálló számítógépre. A virtuális felhasználók a webserververhez csatlakoznak, míg az adatbázis kiszolgáló nem elérhető az internetről, csak belső helyi hálózaton keresztül, ami az előző konfigurációhoz képest biztonság szempontjából is előrelépést jelent.



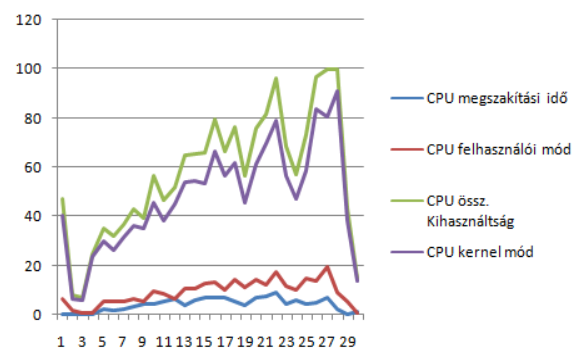
25. ábra – Különálló web és adatbázis szerver

Különböző tesztek futtatva egyértelműen kimutatható volt, hogy a webszerver jobban ki van használva. Míg az adatbázis kiszolgáló CPU terheltségének csúcstértéke csak 40% körüli volt, a web kiszolgálóé elérte a 100%-ot.

Kettéválasztva a funkciókat majdnem megduplázódott a maximálisan kiszolgálható felhasználók száma, mivel a webszerver terheltségének maximumát 150 virtuális felhasználó mellett érte el, ami az egyszerveres konfigurációhoz képest (80 fő) 88%-os javulást jelent.



26. ábra - adatbázis kiszolgáló CPU terheltség

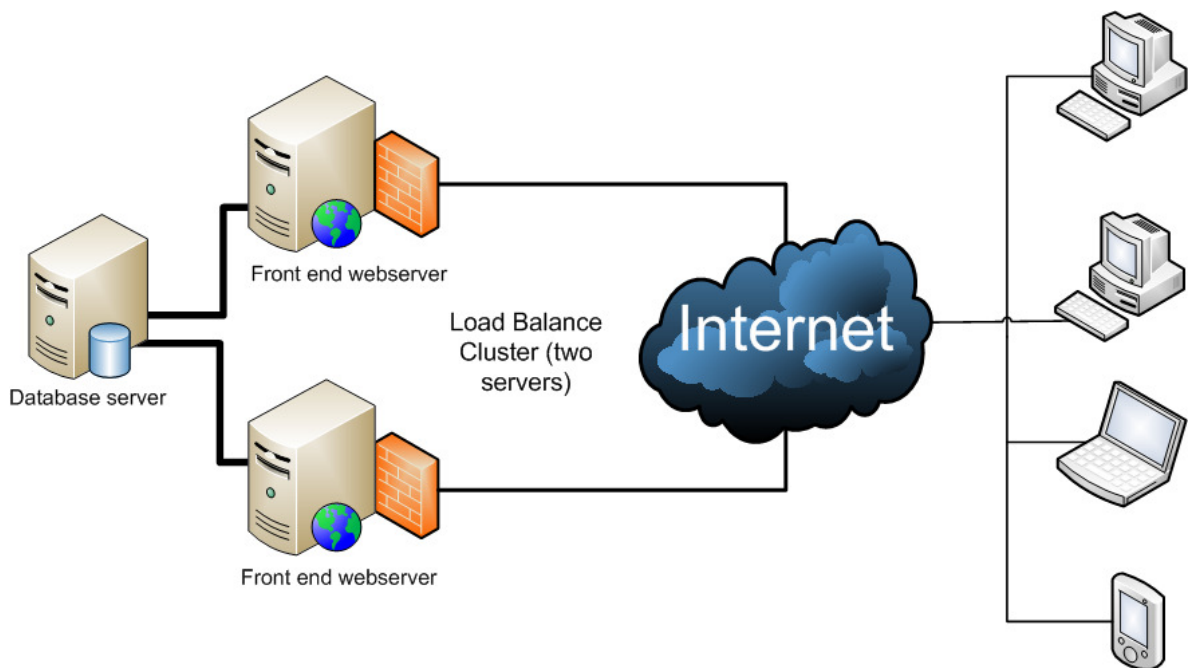


27. ábra - Web kiszolgáló CPU terheltség

Mivel a gépek paraméterei adottak voltak ebben a mérési elrendezésben nem tudtam további méréseket végezni.

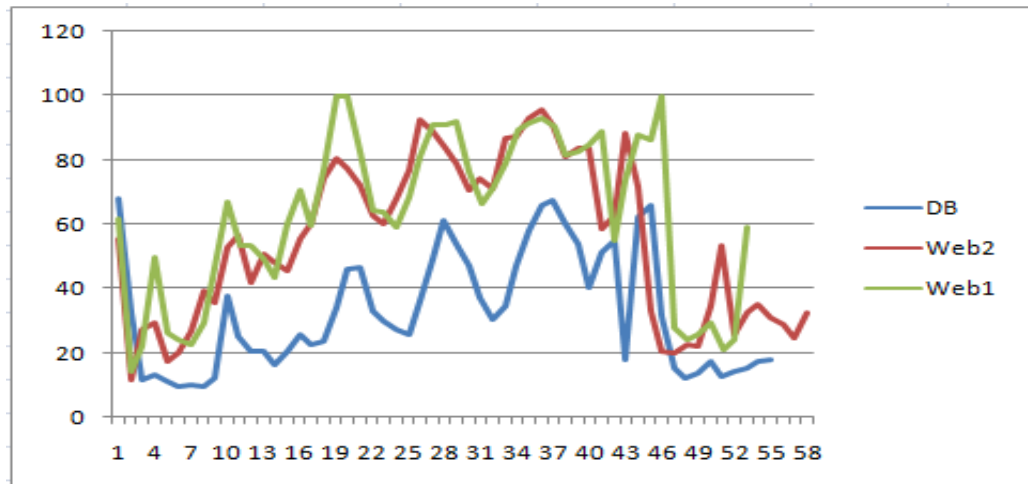
4.2.6 Két webservert terhelés elosztással

Mivel az előző konfiguráció során kiderült, a webservert terheltsége jelenti a szűk keresztmetszetet, a Microsoft Network Load Balancing szolgáltatás segítségével terheléelosztást valósítottam meg. A terheléelosztás automatikusan szétosztja a beérkező kéréseket a webserverek között, így a webserverek számát növelve skálázható rendszer alakítható ki. A két webservert egy közös adatbázis szerverhez csatlakozott.



28. ábra - Két webservert terhelés elosztással

Közös diagramon (29. ábra) ábrázoltam a három számítógép terheltségét egy mérés során. Az adatbázis szerver (kék) terheltsége megduplázódott, terheltsége körülbelül 60-65%-t futott csúcértéken. Ez alatt mindkét webkiszolgáló teljesítménye elérte a 100%-t. Tehát még ebben az esetben is a webservert processzora az a pont, ami megakadályozza további felhasználók kiszolgálását.



29. ábra - Két webservert és az adatbázis szerver CPU kihasználtsága

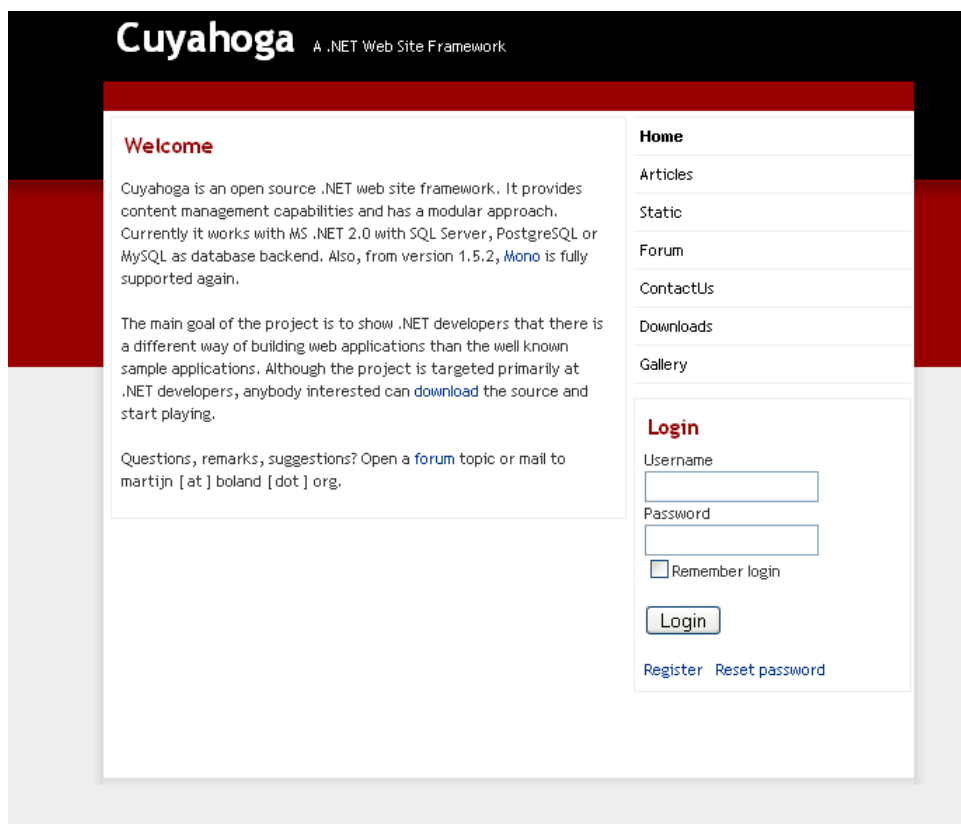
4.3 Teljesítményoptimalizálás fejlesztői szempontból

Fejlesztői szempontból egy összetett webalkalmazást mértem Rational Performance Tester és ANTS Performance Profiler segítségével. Az előbbi segítségével modul szinten, míg az utóbbival forráskód szinten vizsgáltam az alkalmazást.

4.3.1 Tesztkörnyezet meghatározása

A méréseim során a nyílt forráskódú *Cuyahoga* portál keretrendszer teljesítményét vizsgáltam. A méréshez összeállítottam egy teszt portált, amihez a megfelelő modulokat hozzáadtam. A portált feltöltöttem teszt adatokkal, hogy minden modult valós adatok mellett tudjam mérni.

A teljesítméymérést egy egyszerveres környezetben végeztem, a tesztelő és kiszolgáló gép megegyezett. A cél az egyes modulok teljesítményének mérése volt, nem pedig a maximális kiszolgálható felhasználók számának meghatározása. Ennek függvényében öt virtuális felhasználóval játszottam vissza a terhelést.



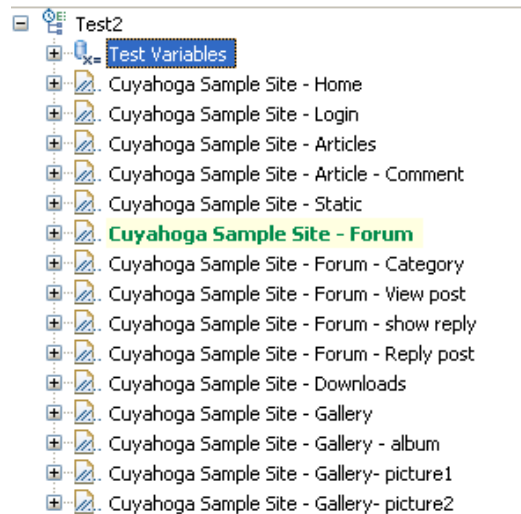
30. ábra – Cuyahoga portál kezdőoldala

4.3.2 Modul szintű teljesítménymérés Performance Tester segítségével

4.3.2.1 Teszteset rögzítése

Olyan tesztesetet állítottam össze, ami a portál összes moduljának funkcionalitását kihasználja, mivel fontos, hogy minden modul teljesítményéről adatot tudjunk gyűjteni.

A teszteset rögzítéséhez létrehoztam egy új projektet a Rational Performance Testerben és hozzáadtam egy új „*Test from recording*” elemet. Ekkor a már előzőekben megismert módon, megjelent egy böngésző, aminek segítségével a tesztesetet rögzítettem, a program pedig a végrehajtott műveleteket rögzítette.



31. ábra – Performance Tester által rögzített teszeset

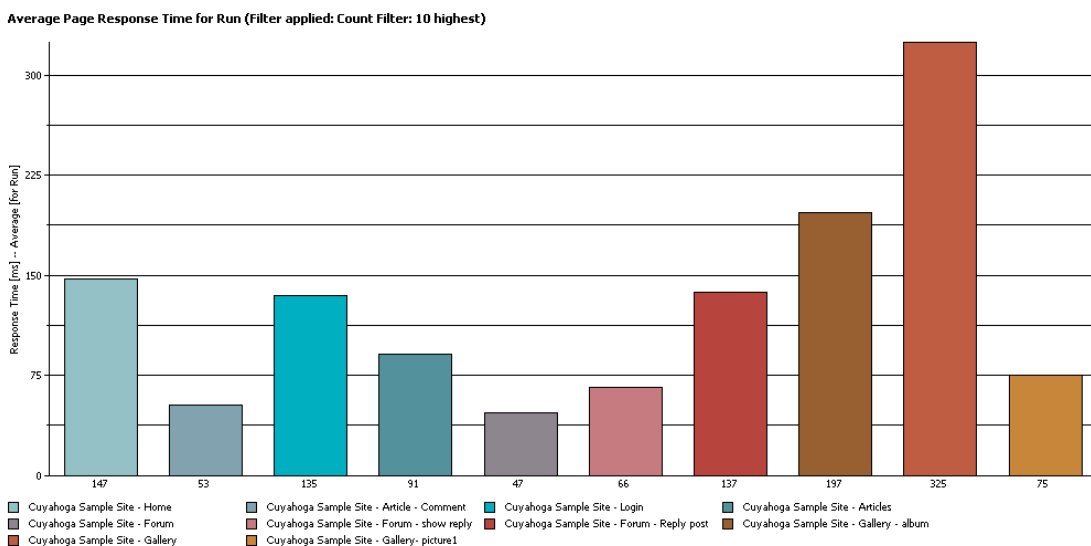
A rögzítést befejezve megjelent egy fa struktúrában minden olyan információ, amit a Performance Tester rögzített. Ekkor lehetőségünk van módosítani, törölni és áthelyezni az egyes lépéseket. Mivel az egyes lépések a megjelenített oldal neve alapján különböztethetők meg, érdemes az azonos oldalon, de más-más műveletet különböző névvel ellátni, hogy későbbiekben könnyebb legyen az azonosítás.

4.3.2.2 Teljesítménymérés

A rögzített teszeset visszajátszásához egy „*Performance Schedule*” elemet adtam hozzá projekthez. A beállításoknál kiválasztottam, hogy előzőekben rögzített teszesetet használja, és öt felhasználót virtuálizáljon. Lehetőségem lett volna teljesítményszámlálók kiválasztására, illetve az IBM Tivoli rendszerekből származó teljesítmény adatok rögzítésére is, de ezekre most nem volt szükségem.

4.3.2.3 Adatok elemzése

A teljesítménymérés befejezése után egy diagramon () megjelent az egyes oldalak (modulok) végrehajtásához szükséges idő.



32. ábra – A Cuyahoga portál moduljainak teljesítménydiagramja

A mérés során a várt eredményeket kaptam. Jól látszik, hogy az egyes modulok teljesítménye más és más. A legtöbb időre a galéria modul megjelenítéséhez volt szükség, ezért a kód alapú elemzés során ezt a modult fogom alaposabban megvizsgálni.

Page Performance

	Response Time [ms] -- Minimum [for Run]	Response Time [ms] -- Average [for Run]	Response Time [ms] -- Standard Deviation [for Run]	Response Time [ms] -- Maximum [for Run]	Attempts -- Rate [per second] [for Run]	Attempts -- Count [for Run]
Cuyahoga Sample Site - Home	125	147	15,66	172	0,13	5
Cuyahoga Sample Site - Articles	47	91	34,85	125	0,13	5
Cuyahoga Sample Site - Login	125	135	8,37	157	0,13	5
Cuyahoga Sample Site - Article - Comment	31	53	13,98	63	0,13	5
Cuyahoga Sample Site - Forum - Category	16	19	6,71	31	0,13	5
Cuyahoga Sample Site - Static	15	37	24,23	62	0,13	5
Cuyahoga Sample Site - Forum	31	47	16,48	63	0,13	5
Cuyahoga Sample Site - Forum - show reply	47	66	10,12	78	0,13	5
Cuyahoga Sample Site - Forum - View post	16	34	17,79	62	0,13	5
Cuyahoga Sample Site - Downloads	15	25	8,69	32	0,13	5
Cuyahoga Sample Site - Forum - Reply post	93	137	43,75	188	0,13	5
Cuyahoga Sample Site - Gallery - picture2	16	25	9,95	32	0,13	5
Cuyahoga Sample Site - Gallery - picture1	31	75	34,7	110	0,13	5
Cuyahoga Sample Site - Gallery	78	325	143,83	453	0,13	5
Cuyahoga Sample Site - Gallery - album	78	197	96,83	344	0,13	5

33. ábra – A Cuyahoga portál moduljainak teljesítménye táblázatos formában

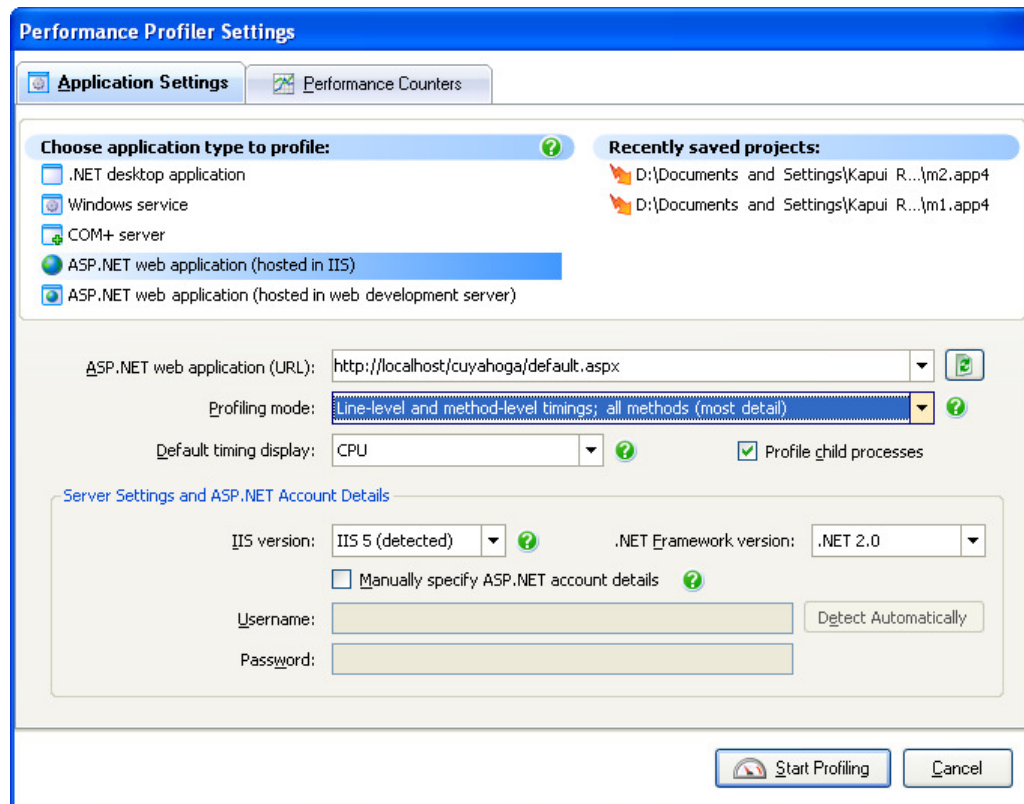
4.3.3 Forráskód szintű teljesítménymérés Performance Profilerrel

4.3.3.1 Teljesítménymérés

Az *ANTS Performance Profiler* segítségével tovább folytatom Cuyahoga portál moduljainak mérését. A profiler futtatása előtt ki kell választani azt az alkalmazást amit mérni szeretnénk. Jelen esetben ez egy webalkalmazás amit IIS-ben (Internet Information Services) hosztolunk. A program három különböző szintű elemzést tesz lehetővé:

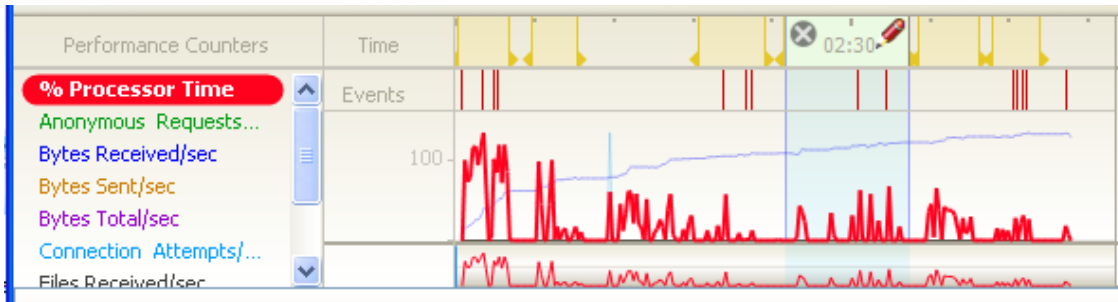
- Csak forrással rendelkező metódus szintű elemzés – leggyorsabb
- Összes metódus elemzése – átlagos
- Metódus és sor szintű elemzés - legrészletesebb

Jelen esetben a legrészletesebb, sor szintű elemzést választottam, hogy a legtöbb információ álljon rendelkezésre. A második fülön kiválaszthatjuk, mely teljesítményszámlálók értékeit rögzítse a program a mérés során.



34. ábra – Az ANTS Performance Profiler beállításai

A rögzítése elindítása után az idővonalon folyamatosan követhetjük a kiválasztott teljesítményszámlálók értékeinek változását, illetve az egyes időszakokhoz címkéket rendelhetünk, hogy később könnyebben lehessen azonosítani.



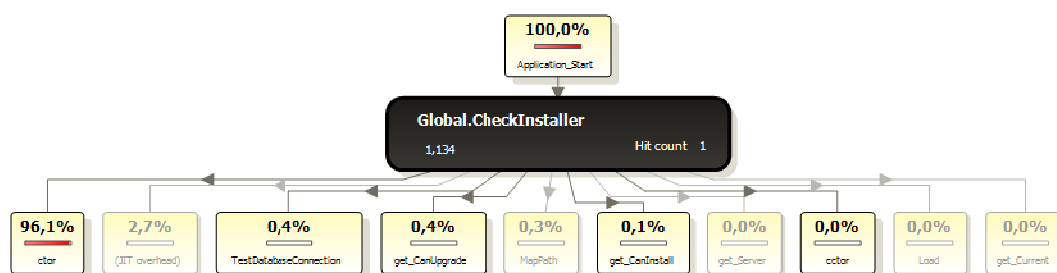
35. ábra – Az ANTS Performance Profiler idővonalán a teljesítményszámlálók

A mérés alatt ugyanazokon a pontokon mentem végig, mint a modul szintű mérés során.

4.3.3.2 Adatok elemzése

Alkalmazás indítása

Az alkalmazás az első oldalt (webszerver indítása utána) feltűnően lassan jeleníti meg. Bár ez az éles futás során nem okoz problémát, megnéztem miért kell 2-3 másodperc az első oldal megjelenítéséhez.



36. ábra –A Cuyahoga webalkalmazás indításának folyamata ANTS Performance Profilerrel ábrázolva

Sorba rendezve a metódusok lefutási idejét kiderült, hogy az Application_Start esemény kezelőjében több mint egy másodpercig várakozik az alkalmazás a CheckInstaller metódusban. A CheckInstaller-t tovább bontva megtaláltam, hogy közel egy másodperc

kell ahhoz, hogy létrejöjjön a kapcsolat az adatbázis kiszolgálóval. Mivel valószínűsíthetőleg ez a legelső kapcsolat, ezért ez normálisnak tekinthető.

Galéria modul elemzése

Megvizsgáltam milyen metódusok futnak le a galéria modul megnyitásakor, hogy kiderítsem, miért lassabb, mint a többi modul. A *PageEngine.OnInit()* metódus az, ami minden oldal betöltésekor lefut, és a *LoadContent()* feladata a megjelenítendő vezérlő létrehozása. A *LoadContent()* metódus lefutási idejét összehasonlítva a többi modul idejével, a galéria modul ideje nagyságrenddel több.

HOT	Cuyahoga.Web.UI.PageEngine.OnInit(EventArgs e)	0,587
HOT	Cuyahoga.Web.UI.PageEngine.LoadContent()	0,433
HOT	Cuyahoga.Web.UI.PageEngine.CreateModuleControlForSection...	0,404
	(Collapsed methods without source, such as framework class library me...	0,029
	Cuyahoga.Web.UI.BaseTemplate.get_Containers()	<0,001
	Cuyahoga.Web.Util.UrlHelper.GetApplicationPath()	<0,001

37. ábra – Galéria modul betöltésének ideje (0,587s)

A galéria modul *OnInit(...)* metódusa 0,587 mp alatt futott le, míg az alábbi ábrán látható letöltések modul 0,082 mp alatt, mely körülbelül hétszeres időkülönbséget jelent.

HOT	Cuyahoga.Web.UI.PageEngine.OnInit(EventArgs e)	0,082
HOT	Cuyahoga.Web.UI.PageEngine.LoadContent()	0,028
	Cuyahoga.Web.UI.PageEngine.LoadMenus()	0,016
	(Collapsed methods without source, such as framework class library methods)	0,015
	Cuyahoga.Core.Service.SiteStructure.SiteService.GetSiteByUrl(string siteUrl)	0,012

38. ábra Letöltések modul betöltésének ideje (0,082s)

Regular Expression

Vizsgálva az egyes metódusok lefutásának idejét, meglepődve tapasztaltam, hogy *UrlHandlerModule.RewriteUrl(string urlToRewrite, HttpContext context)* a tízedik leglassabban lefutó metódus. Míg az előtte álló 9 metódus mind komplex feladatot valósít meg, mint adatbázis kapcsolódás, konfigurációs fájl beolvasása, vagy adatbázis verziójának ellenőrzése, addig a *RewriteUrl* egy regular expression (RegEx) segítségével ellenőrzi, hogy a kapott URL megfelel-e valamely, a konfigurációs állományban beállított *UrlMapping*-nak.

A metódus további elemzésével kiderül, hogy a metódus `Regex.IsMatch(urlToRewrite)` sora meglepően lassan fut le, a metódus futásidejének 80%-át teszi ki. A Cuyahoga portál új verziójában (1.5.2) ezt a hibát már kijavították, ott a méréseim 0,001 mp alatt futottak le.

5 Értékelés

A mérések során sikerült végigmennem azokon az eseteken, amit elterveztem, és azokat az eredményeket kaptam, amit a tervezés során elvártam.

5.1 Teljesítményoptimalizálás felhasználói szempontból

A mérés során kipróbáltam a FWPTT nevű nyílt forráskódú terhelés generáló alkalmazást. A Community Server tartalomkezelő rendszert vizsgáltam egy, illetve 10 konkurens felhasználó mellett. Az eredményen szépen látszott, hogy a válaszidő párhuzamos kiszolgálás esetén lassabb volt, mintha a tíz felhasználó külön-külön hajtotta volna végre a kéréseket.

A mérés során figyeltem, mekkora pontatlanságot jelent az, ha a terhelő és a terhelendő alkalmazás ugyanazon a gépen helyezkedik el. Egy felhasználós esetben nem lehetett megállapítani, mivel a terhelés és kiszolgálás erőforrás igénye nem volt összemérhető a szerver teljesítményével. Tíz felhasználós esetben már körvonalazódott, hogy a terhelésgenerálás a kiszolgáló gép processzor teljesítményének körülbelül 5-6%-át jelentette, és a teljes processzor kihasználtság 10%-át képezte. Ennek megfelelően a mért adatok kevesebb, mint 10% pontatlanságot tartalmazhatnak, ha terhelő és kiszolgáló alkalmazás ugyanazon a gépen helyezkedik el.

A mérési folyamat, amit bemutattam más kisebb alkalmazások esetén is alkalmazható és felhasználható. Bár más alkalmazás mérése esetén a rögzített tesztesetek és az teljesítmény célok eltérnek, a program funkciót hasonlóképpen, csak más paraméterek és beállítások mellett kell használni.

5.2 Teljesítményoptimalizálás üzemeltetői szempontból

Üzemeltetői szempont szerinti optimalizálás során egy forráskód szinten optimalizált mintaalkalmazást mértem úgy, hogy fokozatosan javítottam a mérési elrendezésen, az

egyszeres méreستől a terhelés elosztott webszerverekig, és minden esetben meghatároztam a szűk keresztmetszetet.

Első esetben abba a hibába estem, hogy a kiszolgáló gép nagyságrenddel erősebb volt, mint a terhelő gép. Emellett, egy olyan erőforrás okozta a szűk keresztmetszetet, amire korábban nem is gondoltam volna, a hálózati csatoló. A hálózati eszközt kicserélve, a terhelő gép processzora bizonyult kevésnek a terhelés előállítására, így megakadályozva a konkurens felhasználók további növekedését.

Miután nem tudtam növelni a terhelő gép teljesítményét (és nem állt rendelkezésemre több gép elosztott terhelésgenerálásra), a kiszolgáló teljesítményét vettem vissza.

Továbbiakban a webszerver teljesítménye jelentette a szűk keresztmetszetet, ezért terhelés elosztással megdupláztam, és újabb méreست indítottam. Terhelés elosztás ellenére, a két webszerver is kevésnek bizonyult további kérések kiszolgálására.

További mérések elvégzését nem tartottam szükségesnek, mivel üzemeltető szempontból nem tudtam tovább optimalizálni. Azonos teljesítmények mellett, körülbelül három webkiszolgáló adatbázis lekérdezéseit volt képes kiszolgálni egy adatbázis szerver.

Az üzemeltetői optimalizálás során felhasználtam a vízszintes skálázás több módszerét is. A bemutatott mérési folyamat minden olyan esetben használható, ahol a skálázhatóság ellenőrzése fontos szempont, és meg szeretnék tudni, mekkora az a maximális felhasználó szám és válaszidő, amit egy, kettő vagy több kiszolgáló segítségével biztosítani tudunk.

5.3 Teljesítményoptimalizálás fejlesztői szempontból

A fejlesztői szempont szerinti optimalizálás során először egy nyílt forráskódú portálrendszert mértem meg modul szinten Rational Performance Tester segítségével, majd az egyes modulok forrását is elemeztem az ANTS Performance Profilerrel.

Ahogy a tervezés során feltételeztem, az egyes modulok teljesítménye eltért, nem egyforma idő alatt hajtották végre a kéréseket. A leglassabb modulnak a galéria bizonyult, amit a modul megjelenítésére szolgáló vezérlő okozott.

Előzetesen feltételeztem, hogy az NHibernate használata a teljesítmény rovására fog menni, mivel így az adatbázis kiszolgáló nem tud a lekérdezésekre végrehajtsági tervet készíteni. Ezt megcáfolták a méréseim, ahol az adatok lekérdezése az adatbázisból a kérés kiszolgálásnak a töredékét tette ki. Ebben szerepet játszhatott az is, hogy kevés adatot tartalmazott az adatbázis, illetve a nagy fájlokat és képeket nem a relációs adatbázisban, hanem fájl szinten tárolta a portál.

Viszont meglepően lassú részeket fedeztem fel a forrásban, mint például az Url RegEx vizsgálata. A RegEx kényelmes, egyszerű és komplex szabályokat egyaránt le lehet benne írni, de amint a mérés megmutatta ez a teljesítmény rovására mehet. Nem vizsgáltam, hogy a RegEx általánosságban lassú, vagy csak ebben a speciális környezetben, de ez a teljesítmény egy portál motor részeként, ilyen körülmények között nem elfogadható.

Alkalmazások kód szintű optimalizálásra létezik a legkevesebb úgy nevezett jól bevált gyakorlat. Nehéz minden alkalmazásra érvényes módszert megadni, mivel az egyes alkalmazások nyelvben, architektúrában és platformban eltérnek, ezért az optimalizálásukra használható módszerek és eszközök is speciálisak. Ennek következtében az általam bemutatott módszer is csak .Net alapú webalkalmazások esetén használható.

5.4 Alkalmazott szoftverek összehasonlítása

A tesztelések során sikerült több, webalkalmazás tesztelését és optimalizálását elősegítő alkalmazást megismerem. Törekedtem úgy megválasztani az alkalmazásokat, hogy azok a felső kategóriát és kisebb komplexitású nyílt forráskódú programokat is egyaránt képviseljék. Megismerkedtem két multinacionális cég tesztelő programjával, a Microsoft Visual Studio Team System kiadásával és az IBM cég Rational Performance Tester termékével. Emellett egy szabadon felhasználható, nyílt forráskódú szoftvert is

kipróbáltam, és próbáltam megállapítani, melyek azok az esetek, amikor nincs szükségünk drága szoftverre, hanem például az általam kipróbált FWPTT is megfelel a célnak.

A Visual Studio Team System és Rational Performance Tester hasonló képességekkel rendelkeznek. Mindkét program esetében az általunk választott web böngésző segítségével rögzíthetjük a forgatókönyvet, és a felvett tesztet különböző beállítások mellett játszhatjuk vissza. Mindkét alkalmazás esetén a Windows beépített teljesítményszámlálói rögzíthetjük a mérés során, az IBM terméke továbbá lehetőséget biztosít UNIX rstatd-ből származó adatok rögzítésére, illetve az IBM Tivoli rendszerekhez való kapcsolódásra.

Míg a Microsoft megoldása .NET alapú webalkalmazásokhoz nyújt támogatást, a Rational terméke a Java nyelvet részesíti előnyben, és abban a környezetben nyújt lehetőséget kód analízisre. Mivel funkcionalitásban nem igazán lehet érdemi különbséget tenni ezen a szinten a két program között, a választásnál az dominálhat, hogy a felhasználó a Visual Studio-t vagy Eclipse alapú eszközöket kedveli, ismeri jobban.

Azonban azokban az esetekben, amikor elegendő egyetlen terhelést generáló gép, és nincs szükség elosztott terhelésgenerálásra, az ingyenes nyílt forráskódú eszközök is megfelelhetnek. Bár kezelésük nem olyan kényelmes és egyszerű, mint a multinacionális cégek termékei, használatukkal hasonló eredményt tudunk felmutatni.

Az ANTS Performance Profiler forráskód alapú mérést tesz lehetővé oly módon, amire az előbbi eszközök egyike sem. Amellett, hogy nagyon egyszerű használni, gyorsan és látványosan jeleníti meg a mért adatokat. Hátránya, hogy csak .NET alapú nyelveket támogat, és nincs mód terhelés alatti mérésre sem, mivel nincs beépített terhelés generáló modulja. Méréseim során ez az eszköz nyerte el legjobban a tetszésemet, és használatával tudtam olyan eredményeket felmutatni, amit a többi eszközzel nem tudtam, vagy nem is lehetett volna.

6 Konklúzió

Egy alkalmazás teljesítmény tesztelése és optimalizálása mindig nagy kihívás elé állítja a fejlesztőket, tesztelőket és üzemeltetőket egyaránt. Szerencsére a piacon nagy számban állnak rendelkezésre különböző komplexitású és funkcionalitású programok, amelyekkel ez a folyamat nagymértékben egyszerűsíthető és időben lecsökkenthető.

Fontos, hogy webalkalmazások fejlesztése során a teljesítményt már a kezdeti szakaszban is figyelemmel kell kísérni, teljesítménytesztelési eseteket kell felállítani, és ezeket a szoftverfejlesztés előre haladtával mindig napra készen kell tartani. Nem szabad egyetlen tesztesetre koncentrálni, mivel ahogy a valós életben is, minden felhasználó más és más, így a megszerezni kívánt információ és annak módja is eltér.

Az optimalizáció hiányát csak akkor érezzük, amikor szükségünk lenne rá. De mindaddig, míg nem állnak rendelkezésünkre pontos teljesítmény vizsgálati adatok, nem tudjuk mit is kellene optimalizálni, mi az, ami a teljesítmény csökkenést okozza. Mivel az alkalmazások optimalizálása kezd szerves része lenni a szoftverfejlesztésnek, ennek következtében a fejlettebb fejlesztőeszközök és fejlesztőrendszerek beépített teljesítmény vizsgálati modullal rendelkeznek, így a legtöbb esetben nincs szükség más szoftver beszerzésére és megismerésére.

Mint saját esetemben is kiderült, a szűk keresztmetszetek általában ott vannak, ahol a legkevésbé számítunk rájuk. Az esetek nagy részében a nagy terhelés alatt jelentkező hibákat nehéz később ismételtelen előidézni, így a problémát is nehéz megoldani. Az általam vizsgált webalkalmazások esetében is bebizonyosodott, hogy lineárisan növekvő felhasználó szám mellett a terhelés nem lineáris mértékben növekedett. Egy bizonyos pont felett a teljesítmény nagymértékben visszaesett, és egyre több kérés hiúsult meg, és futott hibára.

A webalkalmazások nagy része relációs adatbázis kiszolgálókra épül, ezen szakdolgozat keretein belül nem foglalkoztam az adatbázis szerverek teljesítmény vizsgálatával. Bár az

adatbázis tervezésére a kezdeti fázisban kerül sor, egy rosszul megtervezett adatbázis séma, illetve adatszerkezet hatására általában csak későn derül fény, olyan későn, amikor már csak nagy költségek és áldozatok árán lehet csak őket kijavítani. Ezért különösen fontos a megtervezett adatbázis mérése, tesztelése és optimalizálása. A téma folytatása lehet az erre vonatkozó ajánlások, módszerek, illetve a rendelkezésre álló programok áttekintése.

7 Irodalomjegyzék

- [1] Daniel A. Menasce, Virgilio A. F. Almeida: *Capacity Planning for Web Services: metrics, models, and methods*, Prentice Hall PTR, ISBN 0130659037, 2001.
- [2] Gyórfi László: *Tömegkiszolgálás informatikai rendszerekben*, Műegyetemi Kiadó, 1996.
- [3] Microsoft Corporation: *patterns & practices: Performance Testing Guidance for Web Applications*, URL: <http://www.codeplex.com/PerfTestingGuide/>
- [4] Nussbaum, D. and Agarwal, A.: *Scalability of parallel machines. Commun. ACM* 34, 3 (Mar. 1991), 57-61. DOI= <http://doi.acm.org/10.1145/102868.102871>
- [5] Microsoft Corporation: *Fundamentals of Engineering for Performance*
URL: <http://msdn.microsoft.com/en-us/library/ms998534.aspx>
- [6] Microsoft Corporation: *Tuning .NET Application Performance*
URL: <http://msdn.microsoft.com/en-us/library/ms998583.aspx>
- [7] Microsoft Corporation: *Visual Studio Team System 2008 Team Suite*
URL: <http://msdn.microsoft.com/en-us/vsts2008/default.aspx>
- [8] *Rational Performance Tester*
URL: <http://www-01.ibm.com/software/awdtools/tester/performance/>
- [9] FWPTT *web load testing framework*, URL: <http://fwptt.sourceforge.net/>
- [10] *ANTS Profiler*, URL: http://www.red-gate.com/Products/ants_profiler/index.htm
- [11] Microsoft Corporation: *Monitoring and Tuning Your Server - Useful Counters for Stress Testing*, URL: http://www.microsoft.com/technet/prodtechnol/windows2000serv/reskit/iisbook/c05_useful_counters_for_stress_testing.msp?mfr=true
- [12] Microsoft Corporation: *Rendszermonitorozás a .NET-ben- Teljesítményszámlálók*
URL: http://download.microsoft.com/download/e/3/d/e3df6fd7-7d24-4e13-a340-8a383d999ffe/TN_2004_05_16-20.pdf
- [13] Microsoft Corporation: *Microsoft Office Excel 2007*
URL: <http://office.microsoft.com/hu-hu/excel/HA101656321038.aspx>
- [14] Wikipedia - *Software performance testing*
URL: http://en.wikipedia.org/wiki/Software_performance_testing
- [15] *.NET Pet Shop 4.0*
URL: <http://blogs.vertigosoftware.com/petshop/archive/2006/02/06/2127.aspx>
- [16] Cuyahoga Web Site Framework, URL: <http://www.cuyahoga-project.org/>