Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Terheléselosztó fürtök és teljesítménytesztelés

Mérési segédlet

Szolgáltatásbiztonságra tervezés labor (BMEVIMIM236)

Készítette: Paljak Gergely, paljak@mit.bme.hu

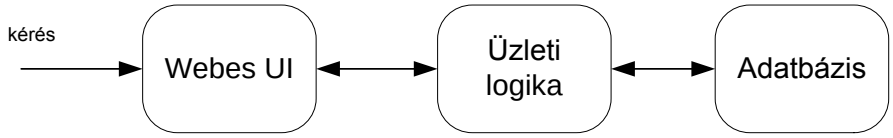
Micskei Zoltán, Gönczy László korábbi útmutatóit, jegyzeteit felhasználva

1 Bevezetés

A modern, elsősorban webes, alkalmazások esetén helyes működés (funkcionalitás) mellett a nem-funkcionális követelmények (szolgáltatásbiztonság, teljesítmény) is egyre fontosabbak. A segédlet a skálázódás, a terheléselosztás és a teljesítménytesztelés alapfogalmait ismerteti, a mérésen ehhez kapcsolódó gyakorlati technológiákat fogunk kipróbálni. A mérési segédlet egy rövid (elméleti) áttekintést ad, a mérés pontos menetét az ott kiosztott mérési feladatok írják le.

1.1 Háromrétegű architektúrák

Webes alkalmazások megvalósítása esetén gyakran használt felépítés a három (vagy finomabb granularitású rendszer esetén az N) rétegű architektúra. Logikai szinten elkülönítjük a főbb funkciókat ellátó rétegeket, úgymint: web, üzleti logika, adatbázis (1. ábra).



1. ábra: Egy háromrétegű architektúra

Hasonló rétegezést majd minden manapság használatos platformon létre tudunk hozni, a mostani mérés folyamán nyílt forráskódú technológiákat fogunk alkalmazni.

1.2 A skálázhatóság

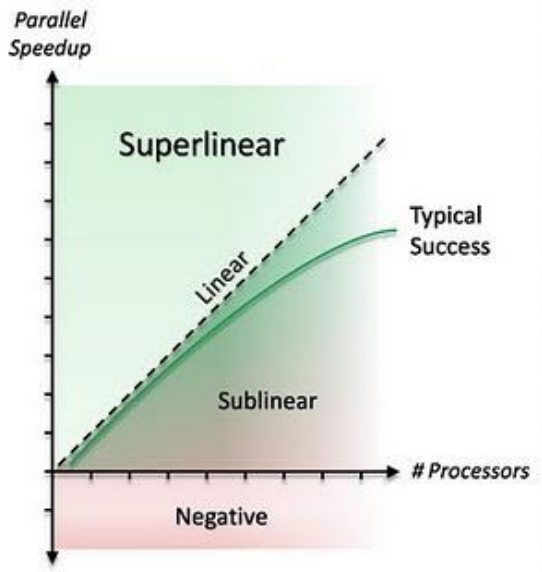
A skálázhatóság a rendszer azon tulajdonsága, hogy a rendszer teljesítménye a rendelkezésre álló erőforrások változtatásával milyen módon változik. Definiáljuk most a teljesítményt a rendszer maximális áteresztőképességével a processzorok számának függvényében, legyen ez: $X_{\max}(p)$, ahol p a processzorok száma. Legyen ez az alapján a skálázódási faktor $C(p) = \frac{X_{max}(p)}{X_{max}(1)}$. Természetesen más metrikát is választhattunk volna, nem csupán az áteresztőképességet, és ahhoz is mérhetnénk a skálázódást; a teljesítménymetrikákkal a 3.4 fejezet foglalkozik.

A 2. ábra a skálázást jellemző legfontosabb fogalmakat mutatja be.

Ha például egy skálázható rendszerben kétszeresére növeljük a processzorok számát, akkor úgy véljük, hogy a kiszolgálható felhasználók száma is közel a kétszeresére növekszik. Ez nyilvánvalónak tűnik, ugyanakkor egy valós rendszerre nem feltétlenül igaz, csak „kivételes", *lineárisan* skálázható rendszerekre, ilyenkor $C(p) = p$. A legtöbb valós rendszer azonban ennél rosszabb, *szublineáris* *($C(p) < p$)*, vagyis az erőforrások kétszeres megnövelése nem növeli kétszeresen a teljesítményt (szűk keresztmetszet, kiegyensúlyozatlanság miatt). Nagyon ritkán beszélhetünk *szuperlineáris* skálázódásról ($C(p) > p$), amikor is a teljesítmény a lineárisnál nagyobb mértékben nő. [12]

Egy adott alkalmazás skálázására alkalmas módszereket két kategóriába sorolhatjuk, a vertikális skálázásra (*scale vertically, scale up*) és a horizontális skálázásra (*scale horizontally, scale out*). A vertikális skálázás során a rendszer egy kiválasztott eleméhez adunk új erőforrást, tipikusan egy számítógépet bővítünk processzorral, memóriával vagy háttértárral. A horizontális skálázás esetén a rendszert bővítjük egy új elemmel, például számítógéppel vagy routerrel. Skálázás eszköze lehet például a terhelés elosztás

(*load balacing*), 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.



2. ábra Informatikai rendszerek skálázása

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

- A felhasznált hardver rosszul 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 korlátos memóriaterületet tud használni,
- Nagy memória-igényű alkalmazások nem fordítanak elegendő figyelmet a nem használt memória felszabadítására.

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

2 Terheléselosztás

A terheléselosztás alapvető célja inkább a teljesítménynövelés, skálázódás biztosítása, de természetesen a redundancia miatt hibatűrést is biztosít. Az egymástól független gépeket egy fürtbe szervezzük, ami kívülről egy közös (gyakran virtuális) címen érhető el. Az erre a címre érkező kérések szétosztása a csomópontok között az egyes megvalósításokban egész eltérő lehet. A legegyszerűbb megoldásokban

egy statikus lista alapján működik ez (pl. round-robin DNS), míg a legkifinomultabb esetben a csomópontok folyamatosan kommunikálnak egymással, hogy felmérjék a másik aktuális terheltségét és figyelik, hogy az adott klienst ki szolgálta ki az előző kérésénél.

2.1 A terheléselosztás céljai

A terheléselosztás alapvető célja a beérkező kérések elosztása a külvilág felé egyetlen kiszolgálónak látszó kiszolgálók csoportja (fürtje) között. Fontos jellemzője, hogy a kérések elosztását önálló, a kérések kielégítésére önmagukban is képes, független kiszolgálók között végzi, illetve az elosztás művelete a hálózati rétegben történik, szemben például az alkalmazások, az alkalmazáskomponensek vagy az objektumok közötti elosztással.

Ezzel szemben a feladatátvételi fürtök célja elsődlegesen a rendelkezésre állás növelése. A fürt által nyújtott szolgáltatást általában egy csomópont nyújtja egyszerre, ennek meghibásodása esetén a többiek közül valaki átveszi a szerepét, így csökkentve a szolgáltatás kiesését. A feladatátvételi fürtökkel a következő mérés foglalkozik.

- Állandó jellegű adatokkal dolgozó kiszolgálók (FTP, web) esetében alkalmazható hálózati terheléselosztás.
- Változó adatok kezelésére (pl. adatbázis, levelezés) a közös tárolóeszközt használó feladatátvételi fürtök az alkalmasabbak.
- A fürtözés a kiszolgálói alkalmazás megváltoztatása nélkül történik, a kiszolgáló alkalmazás nem tud arról, hogy őt fürtözik.

A terheléselosztásnál a technológia kiválasztásának alapkérdése, hogy vajon alkalmazzunk-e egy kevésbé egyenletes terheléselosztást eredményező, de egyszerű módszert, vagy egy minél kifinomultabban szabályozott, minél egyenletesebb elosztást biztosító, de magas többletterheléssel járót.

2.2 Tipikus terheléselosztási topológiák

Ebben az alfejezetben a jól ismert terheléselosztási topológiákat, mintákat ismertetjük.

2.2.1 Dedikált szerver

Egyetlen dedikált szervert vagy hardver terheléselosztót használunk, amelynek kizárólagos feladata a terheléselosztás. Előnye, hogy olcsó megoldás (kis teljesítményű hardver is megfelel erre a célra), nincs szinkronizáció (hiszen csak egy terheléselosztó elemet használunk). Viszont nem hibatűrő, mert egyetlen meghibásodás a szolgáltatás kieséséhez vezet.

2.2.2 Colocated szerver

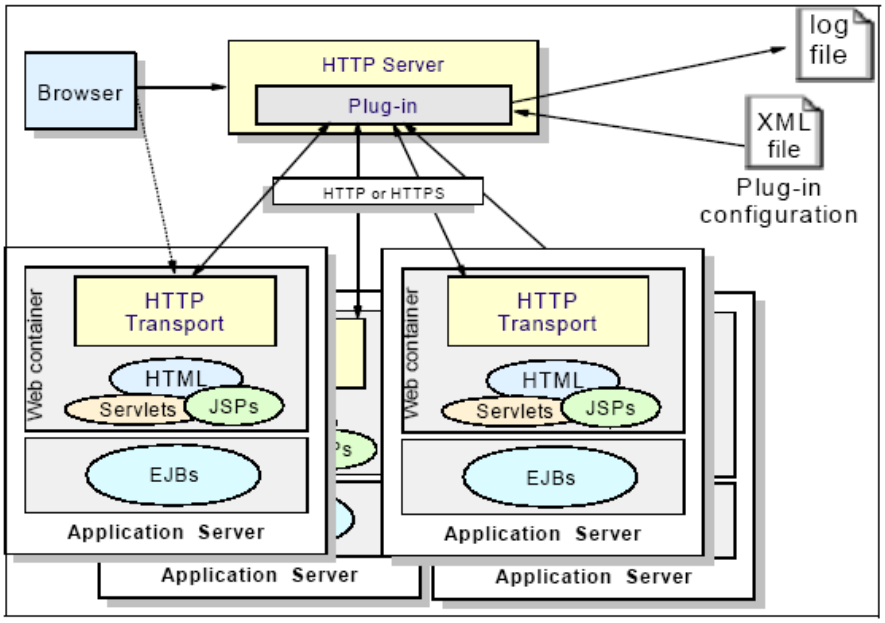
Lehetőség van a terheléselosztót a webszerverrel azonos gépen is elhelyezni (colocation). Tipikusan ezek a gépek kiszolgálják a statikus tartalmat, és a terheléselosztást a dinamikus elemeket kiszolgáló alkalmazás szerverek között végzik. Ez szintén egy olcsó megoldás, mivel a Total Cost of Ownership (TCO, teljeskörű birtoklási költség) még kevesebb, gyakran elegendő egy plug-in installálása. Hátránya, hogy egy gép kiesése a terheléselosztás és a statikus tartalmak kieséséhez vezet, amelyet általában nehezebb javítani, mint egy terheléselosztó meghibásodását. (3. ábra)

2.2.3 Magas rendelkezésre állású terheléselosztás, passzív replikációval

A magas rendelkezésre állású terheléselosztás két terheléselosztóból áll: egy aktív és egy passzív elemből. Egyszerre csak az egyik nyújt szolgáltatást, de a konfigurációt az aktív folyamatosan replikálja a passzív elemre, ezért ez folyamatosan konzisztens. Az egységek folyamatos heartbeat üzeneteket küldenek, hogy tudassák, hogy működőképesek. A heartbeat üzenetek kimaradása meghibásodást jelent, ekkor a passzív egység átveszi az aktív szerepét a kiszolgálásban. Ez a módszer meghibásodás esetén is nagyon rövid szolgáltatás-kimaradást eredményez. Természetesen könnyen kivitelezhető több passzív egység használata is, ekkor ennek megfelelően több meghibásodást is képes elviselni a rendszer. Ez a módszer nyilvánvalóan költségesebb az előzőeknél, hiszen több terheléselosztóra van szükség, amelyeknek ráadásul intelligensebbnek is kell lenniük (heartbeat üzenetek feldolgozása, replikáció és feladatátvétel).

2.2.4 Magas rendelkezésre állású terheléselosztás, aktív replikációval

Hasonló az előző módszerhez, itt is több terheléselosztót használunk, viszont nincs passzív, folyamatosan „pihenő” elem. Hibamentes működés esetén minden terheléselosztó egy-egy fürtöt szolgál ki, viszont képes átvenni egy vagy több másik terheléselosztó feladatát is, ezek konfigurációját folyamatosan replikálja. Más szavakkal: a terheléselosztók egymás tartalékai és képesek automatikusan átvenni egymás feladatait. Látható, hogy ez a megoldás a hardverelemek szintjén kevesebb költséggel jár, viszont még intelligensebb eszközöket igényel (folyamatos replikáció, feladatátvétel)



3. ábra Terheléselosztás egy elosztott, J2EE rendszerben

2.3 A terheléselosztás algoritmusai

A terheléselosztó feladata kijelölni, hogy egy beérkező kérés melyik (kiszolgálására képes) szerverhez jusson el. Ezen feladat megoldását nevezzük a terheléselosztás algoritmusának. A legelterjedtebb megoldások:

Round robin:	A szerverek egy listában vannak, az új kérést a listán következő szerver szolgálja ki, és léptetjük eggyel a mutatót.
Súlyozott round robin*:*	Minden szerverhez rendelünk egy súlyt (tipikusan egy egész szám), a szerverek a súlyok arányában kapják a kéréseket. A súlyok lehetnek előre megadottak (statikus) vagy működés közben számítottak (dinamikus).
Több osztályos round robin:	A különféle terhelésosztályokat figyelembe veszi, pl. hálózat-intenzív – nagy fájlok, I/O-intenzív – adatbázis lekérdezés, CPU-intenzív – biztonságos kommunikáció, kevés terhelést jelentő – kis statikus tartalom. Az osztályokat rendeljük RR (round robin) szerint szerverekhez.
Véletlen:	Minden kérés egy véletlenül választott szerverhez kerül.
Forrás IP hash:	A kérések a forrás IP cím alapján kerülnek a kiszolgálókhoz. Ha meghibásodás történik, akkor meg kell változtatni az elosztó táblát. Hibamentes kiszolgálás esetén garantálja, hogy egy kliens kéréseit mindig ugyanaz a szerver szolgálja ki.
URL hash:	Itt az URL alapján osztja ki a terheléselosztó a kéréseket. Különösen a proxy cache-ek előtti terheléselosztás esetén hasznos, mivel az ugyanarra a címre hivatkozó kérések ugyanahhoz a cache-hez kerülnek. Ami nyilvánvalóan javít a rendszer teljes teljesítményén, hiszen el lehet kerülni a duplikált adatokat a cache-ekben, tehát ugyanannyi cache elem összesen több adatat tárol.
Legkevesebb kapcsolat:	A terheléselosztó nyilvántartja a szerverenkénti nyitott kapcsolatok számát és az új kérést a legkevesebb kapcsolattal rendelkező szerverhez irányítja. (Ennek az algoritmusnak könnyen elképzelhetjük súlyozott változatát is.)
Legkevesebb forgalom:	A terheléselosztó nyilvántartja a szerverenkénti hálózati forgalmat és az új kérést a legkevesebb forgalommal rendelkező szerverhez irányítja. (Ennek az algoritmusnak könnyen elképzelhetjük súlyozott változatát is.)
Legkisebb késleltetés:	A terheléselosztó küld egy gyors lekérdezést (pl. HTTP OPTIONS), és az először válaszoló szerver kapja a kérést.

2.4 Session kezelés

A session vagy munkamenet két szoftver entitás (tipikusan böngésző és web szerver) közötti kommunikáció, mely során az egyik (vagy mindkét) fél átmenetileg adatokat tárol a másikról. A session informatikai rendszerekben lehet állapottal rendelkező vagy nem rendelkező (stateful, stateless). Állapottal nem rendelkező sessionök esetén könnyebb a feladat, hiszen a kérés kiszolgáláshoz nincs szükség korábbi, valamelyik szerveren tárolt információra, így ez nem korlátozza a terheléselosztást.

Az állapottal rendelkező sessionök kezelése és a terheléselosztás összehangolása egy nehézkes, általában kompromisszumokat igénylő feladat. A következő módszerek a leggyakrabban használtak:

- **Session affinity:** egy session csak egy gépen tárolódik, a kliens minden lekérdezését ugyanaz a szerver szolgálja ki. Egyszerűen megvalósítható, ugyanakkor nem hibatűrő, és egyenetlen terheléshez vezethet.

- **Session perzisztencia:** a szerverek időnként lementik a session állapotát egy közösen elérhető tárba (pl. SQL szerver), így tetszőleges szerver tudja folytatni a kliens kiszolgálását. Előnye, hogy hibatűrő, ugyanakkor nehezebb megvalósítani és a válaszidőt megnyújthatja a session mentése és betöltése.
- **Kliensoldali session tárolás:** a sessiont a kliens oldalán is tárolhatjuk, erre három lehetőség közismert:
    - ***Cookie:*** A cookie *(némely magyar forrásban: süti)* egy olyan információ, amelyet a szerver a válaszban elküldhet a kliensnek, hogy az tárolja. Ezután a kliens, amikor a HTTP kérést összeállítja, minden korábbi cookie-t elküld, amelyet a megfelelő szolgáltatástól kapott.
    - **URL újraírás (URL rewriting):** a szerver a kliens oldali állapotot a HTTP GET kérés argumentumaiban kapja meg (így alakulnak ki a rendkívül hosszú, emberek számára általában nem értelmes URL-ek).
    - **Rejtett űrlap változók:** Az <INPUT TYPE="HIDDEN" ... > HTML kifejezés segítségével lehet rejtett mezőket létrehozni, ezekben adatot tárolni. Az ilyen mezők értékeit tartalmazni fogják a későbbi HTML kérések.

3 Teljesítménytesztelés és optimalizálás

Az alábbi fejezet elkészítéséhez felhasználtuk Kapui Ákos „Webalkalmazások teljesítményvizsgálata és skálázhatósága” (konzulens: Micskei Zoltán) című BME BSc szakdolgozatát. [2]

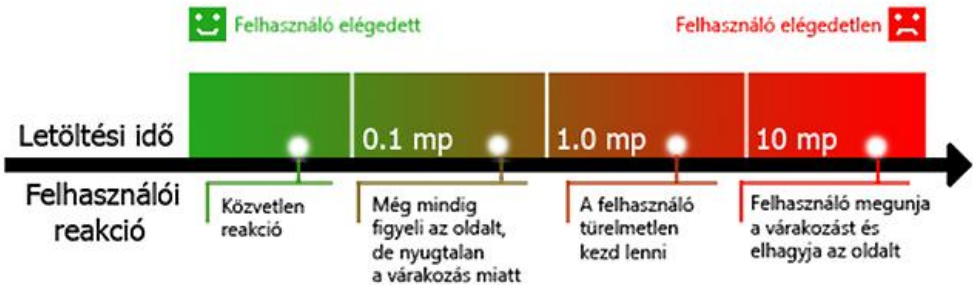
3.1 A teljesítmény definiálása

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. Bármikor, amikor alkalmazást tervezünk, fejlesztünk, tesztelünk vagy menedzselünk, figyelembe kell vennünk a teljesítményt.

A teljesítmény a fejlesztésben résztvevő különböző szereplőkre különbözőképpen hat:

- **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 hozzákezdeni, hol kell folytatni és mikor kell befejezni az optimalizálást.
- **Tesztelőként** ellenőrizni kell, hogy az alkalmazás képes-e az előírt 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.

A teljesítményoptimalizálás egy üzleti szempontból is kritikus feladat, különösen igaz ez a webalkalmazásokra, weboldalakra, ahol néhány másodperccel megnövekedett válaszidő is (lehetséges) ügyfelek elvesztését eredményezheti (4. ábra). Nem véletlen pl. a Google (és sok más nagy szoftvercég) rengeteg ezzel kapcsolatos erőfeszítése.[10]



4. ábra Milyen elvárásaink vannak webalkalmazások teljesítményével szemben?

3.2 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.

3.2.1 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, amely 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.

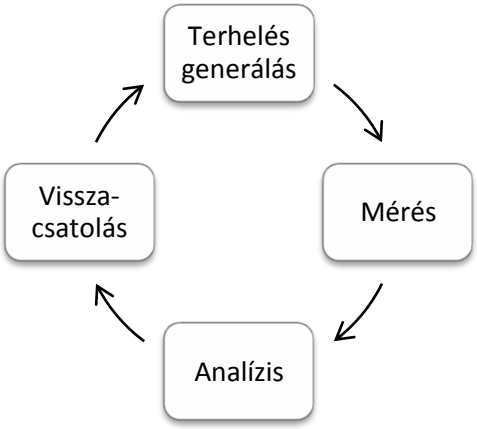
A legtöbb esetben az alkalmazás szűk keresztmetszetét valamely hardver egység teljesítménybeli hiányossága okozza, mint például CPU, memória, merevlemez, hálózati interfész vagy más külső erőforrások (mint pl. az adatbázis kapcsolatok száma vagy a hálózat sávszélessége).

3.2.2 A teljesítményoptimalizálás folyamata

A szoftvertervezésben a 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.

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.



5. ábra A teljesítménytesztelés folyamata

3.2.2.1 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.

3.2.2.2 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.

3.2.2.3 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. Eredménynek számít az is, ha nem találtunk olyan hardverelemet, mely egyértelműen a szűk keresztmetszetet jelenti, sokszor a mérés elemzése komponensek kizárást eredményezi, így más komponenseket tudunk mélyebben vizsgálni.

3.2.3 Teljesítményoptimalizálási eszközök

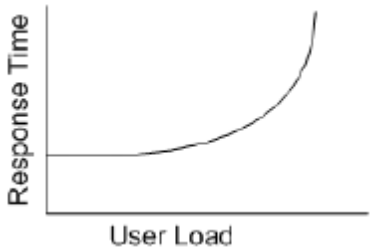
A mérési folyamat egyes részfeladataira különböző cégek eszközei állnak rendelkezésünkre. Minden eszköznek megvannak az előnyei és hátrányai is, találunk köztük egyaránt ingyenes eszközöket és

nehezen megfizethetőeket is. Mindig a feladat komplexitása függvényében döntsünk, mely programokat választjuk a méréseink során.

Gyakran, főleg egyszerűbb rendszerek esetén jó eredményeket lehet elérni bevált gyakorlatok átvételével, erre egy gyűjtemény található [4]-ben. Ugyanakkor ezeknek az egyszerűen alkalmazható szabályoknak a hasznossága sokszor kétséges, nem bizonyítható; szélsőséges esetekre nehezen lehet felkészülni, és a rendszer működésének megértése nélkül kevés sikerrel kecsegtetnek.

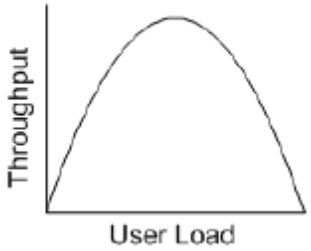
3.2.4 Különböző mérési módszerek ismertetése

Teljesítménymméréshez több, általánosan elfogadott módszer létezik, melyek segítségével több szempontból tudjuk vizsgálni a rendszerünket.



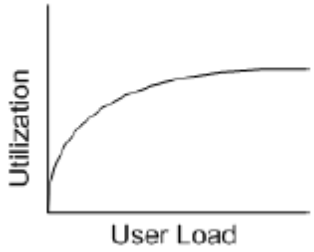
6. ábra Válaszidő és a felhasználók számának kapcsolata

Amikor a válaszidőket vizsgáljuk (6. ábra) 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.



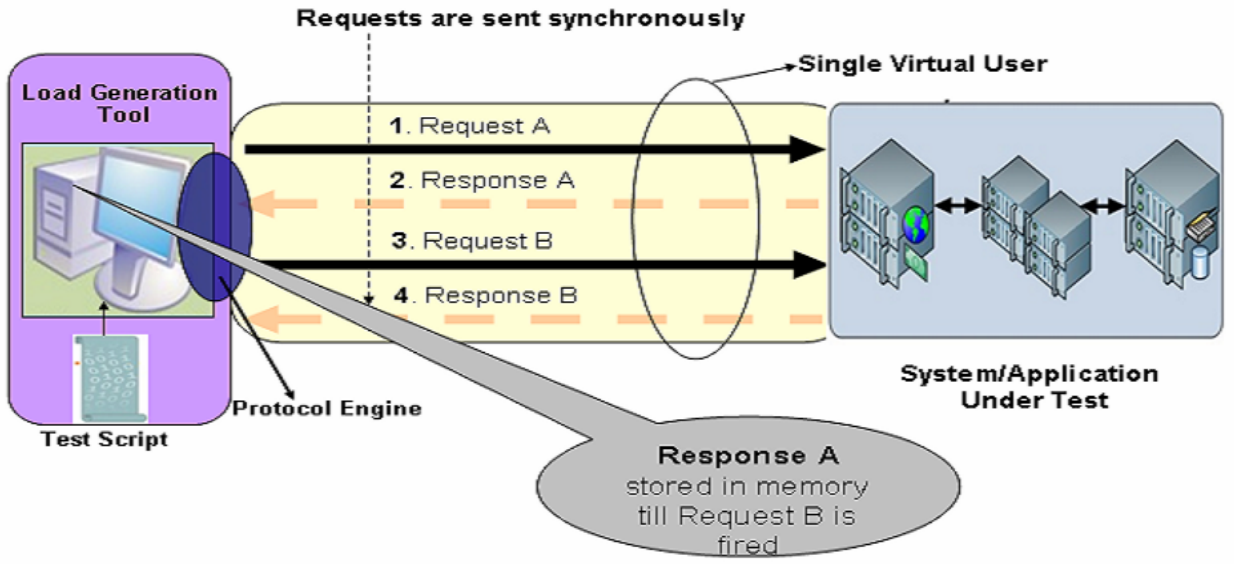
7. ábra Áteresztőképesség vizsgálata terhelés változtatásával

Amikor az áteresztőképességet vizsgáljuk (7. ábra) növekvő terhelés mellett, azt a pontot kell keresnünk, ahol az áteresztőképesség csökkeni kezd. Ezen a ponton elértünk egy szűk keresztmetszetet, további növelés mellett már csökkeni fog a teljesítmény.



8. ábra Erőforrások terheltségének vizsgálata terhelés változtatásával

Vizsgálhatjuk, hogy mikor érik el az egyes erőforrások a 100%-os vagy közel 100%-os kihasználtságot (8. ábra), úgy, hogy a felhasználók számát és ezzel az összeterhelést fokozatosan növeljük.



9. ábra Teljesítménytesztelés, forrás: [6]

3.3 Teljesítménytesztelés áttekintése

A teljesítménytesztelés a szoftvertesztelés egyik részfeladata, szerepe a szoftver teljesítményének vizsgálata: mekkora terhelést, milyen minőségben képes kiszolgálni. Ebben a segédletben a web alkalmazások teljesítménytesztelésének gyakorlati oldalát tekintjük át. Az elméleti háttér bővebb megismeréséhez ajánljuk például az alábbi jegyzeteket: [8][9].

Web alkalmazások teljesítménytesztelése során egy integrált rendszert vizsgálunk (ezt System Under Test-nek, SUT-nek nevezzük): azt akarjuk megtudni, hogy mekkora terhelést képes még megfelelően kiszolgálni, melyik alrendszer a leggyengébb (szűk keresztmetszet), ill. hogyan viselkedik a túlterhelt rendszer (9. ábra).

Az ipari gyakorlatban, komoly szoftverek esetén az éles rendszer mellett általában működik egy tesztrendszer is, amelyen az új komponensek funkcionális-, teljesítmény- és szolgáltatásbiztonsági

tesztelését el lehet végezni. Ez egy igen költséges fázis, hiszen a környezetet exkluzívan kell használni (többféle teszt párhuzamos futása meghamisíthatja az eredményeket), valós körülményeket kell teremteni (több ezer felhasználó szimulálása), amelyhez realisztikus infrastruktúra szükséges. Ezért különösen fontos a megfelelő eszköz(ök) kiválasztása, a gondos teszttervezés, a helyes kiértékelés, majd e tesztek eredményei alapján a megfelelő változtatások végrehajtása.

A teljesítménytesztelés (performance test) sokféle feladatból áll, néhány közismert teszttípust ismertetünk itt:

- **Terheléstesztelés (load test):** a teljesítmény vizsgálata egy előre definiált terhelés mellett, vagyis megmérjük, hogy mennyi ideig tart egyes feladatok végrehajtása előre tervezett, normál működési körülmények között
- **Stressztesztelés (stress test):** a rendszer vizsgálata, amikor az előre tervezett működési tartományon kívül üzemel, pl. valamiféle túlterhelés alatt.
- **Bemelegítés:** Pár konkurens szállal és rövid lefutással ellenőrizzük kialakított stressztesztelő környezetünk működését.
- **Csúcsterhelés (peak test):** Akciók, új információk, különleges alkalmak esetén előfordul, hogy a weboldal látogatottsága hirtelen többszörösére ugrik, és ezáltal a webszerverek, hálózat terheltsége is drasztikusan megváltozik. Ezekre az eseményekre is nagyon fontos felkészülni, hiszen az akciónk fulladhat kudarcba, ha a rendszerünk nincs rá felkészülve. Ezekkel a csúcsterheltségi adatokkal is futtassunk tesztet a rendszeren.
- **Ramp teszt:** Egy fokozatos terhelés növelés során figyeljük a rendszer viselkedését. A teszt során egy lépcsős függvény alapján növeljük a konkurens felhasználók és lapletöltések számát. Egy-egy növelés után több percig kivárunk, hogy az adott konkurencia számhoz tartozó stabil rendszerműködés értékeit kaphassuk meg. Hogy meddig növeljük a konkurens szálak számát, az döntés kérdése. Akár a rendszer teljes összeomlásáig elmehetünk. Ez a teszt tipikusan hosszú lefutású, akár egy napig is eltarthat.
- **Helyreállási teszt (recovery test):** Egy csúcsterhelés után is érdemes a rendszert vizsgálni, hogy áll vissza a normál működési állapotra, képes-e felszabadítani erőforrásait, képes-e a processzor és a hálózat terhelése visszatérni az általános terhelés során tapasztalt értékintervallumba. [3][5][11]

3.4 Teljesítmény metrikák

Mit jelent a metrika? A metrika egy kiszámítható mérőszám, amely egy előre meghatározott halmazból veszi fel értékét, rendezési reláció van az értékek közt (pl. az áteresztőképesség valós szám, a veszélyességi szint enumeráció), időben behatárolt, rendszer elemhez (entitáshoz) rendelt. A teljesítmény metrikák természetesen a teljesítmény egy aspektusát jellemző ilyen merőszámok. [5] [11]

Általános metrikák:

- **Válaszidő:** a kérés első byte-jának elküldése és a válasza utolsó byte-jának beérkezése között eltelt idő (a tranzakció kiszolgálása, a hálózati átvitellel együtt). [s]
- **Áteresztőképesség:** adott idő alatt kiszolgált kérések száma. [1/s]
- **Kihasználtság** (CPU, diszk, memória, hálózat esetén) [%]

Elosztott rendszerek gyakran használt metrikái:

- **Connection time:** a kliens és szerver közötti kapcsolat felállításához szükséges idő. [s]
- **Send time:** a kliensről a szerverre adatok küldéséhez szükséges idő. [s]
- **Receive time:** a szerverről a kliensre adatok küldéséhez szükséges idő. [s]
- **Process time:** a szerveren a kérés kiszolgálásához szükséges idő, a szerveren töltött idő. [s]
- **Response time:** egy tranzakció kiszolgálásához szükséges idő (egy vagy több process time-ot tartalmazhat, hálózati késleltetést és esetleg egyéb késleltetéseket tartalmazhat). [s]
- **Failed Transactions per Second**: a hibát jelző tranzakciók száma másodpercenként (elsősorban hibakeresés céljára). [1/s]
- **Requests per second:** a szervert elérő lekérdezések száma másodpercenként. [1/s]
- **Concurrent connections:** időegység alatt fenntartott párhuzamos kapcsolatok száma. [1/s]
- **Áteresztőképesség:** időegység alatt feldolgozott tranzakció vagy adatmennyiség (egyaránt használatos tranzakció/s vagy byte/s mérőszámmal, a pontos definíció az adott feladattól függ). [1/s]

Webalkalmazás-specifikus metrikák:

- **Hits/sec:** az oldalról letöltött objektumok száma (képek, bannerek is). [1/s]
- **Page View/Day:** adott oldalt hányszor nézték meg. [1/day]
- **Click-throughs:** hányan néztek meg egy adott hirdetést. [1/s]
- **Unique Visitors:** hány különböző látogató volt egy adott időszakban. [1/s]

Webalkalmazás-specifikus, üzleti célokat is figyelembe vevő metrikák

- **Revenue Throughput:** az oldal által elért átlagos bevétel. [$/trans]
- **Potential Loss Throughput:** mennyibe kerül a szolgáltatás kiesése egy adott időszakban
- **Visit Ratio:** átlagosan hányszor vesznek igénybe egy adott szolgáltatást (egy session alatt). [$/s]
- **Buy to Visit Ratio:** átlagosan hányszor vásárolnak egy session alatt (tényleges eladási tranzakció)
- **Average Session Length:** egy session átlagosan hány szolgáltatást vesz igénybe (nem időt mér). [1/session]

Természetesen az itt felsorolt metrikák halmaza messze nem teljes. Egy-egy konkrét feladat az elérendő célok függvényében kívánja a metrikarendszer kidolgozását; erre a feladatra sok módszertan létezik egy közülük az eredetileg a NASA-nál fejlesztett GQM (Goal-Question-Metric) [7].

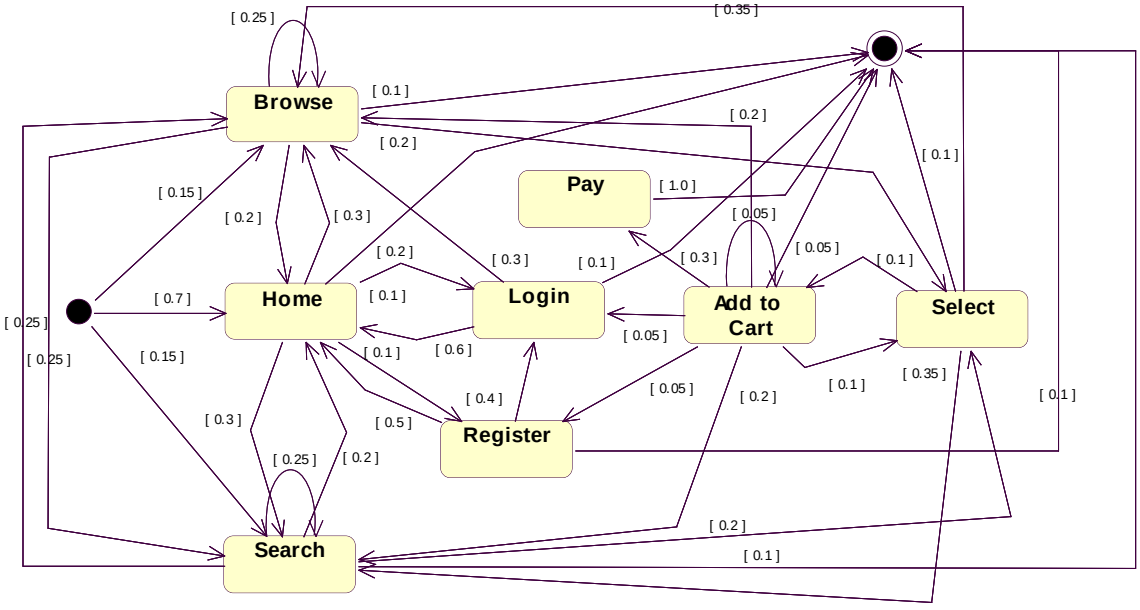
3.5 A felhasználó modellezése

A web alkalmazások realisztikus terhelés modellezéséhez a rendszer felhasználóját, a felhasználó viselkedését kell modellezni, ennek egy szabványos módja a Customer Behavior Model Statechart (CBMS, vásárlói viselkedés állapotdiagram modellek) (10. ábra).

A CBMS egy irányított gráf az oldalak közti lehetséges átmenetek és valószínűségeik ábrázolására. N db csúcsból áll, ahol

- az 1. csúcs az Entry állapot, absztrakt belépési pont, minden felhasználó innen indul, ide nem tér vissza,
- az n. csúcs pedig az Exit állapot, absztrakt kilépési pont,
- a többi csúcs megfelel a felhasználó által elérhető szolgáltatásoknak (oldalaknak),
- az élek: az oldalak szerkezete határozza meg a csúcsok közt lehetséges átmeneteket.

Ahhoz, hogy ezt a gráfot fel tudjuk írni, az első lépés az oldalak által nyújtott szolgáltatások meghatározása (pl. Login, Add Item, Get Quotes), majd szükséges a szolgáltatások halmazának finomítása – az infrastruktúrát különböző mértékben terhelő szolgáltatásokra (pl. Download szétválasztása Download Audio, Download Video szolgáltatásokra), végül a lehetséges átmenetek meghatározása – az oldalak megjelenítésének vizsgálatával (linkek, formok, menüpontok). A lehetséges átmenetekhez valószínűségi súlyokat kell rendelni, ezzel növelve a modell valósághűségét: a súlyokat célszerű mérések alapján (pl. logokban gyakoriság elemzése) meghatározni, egyébként csupán találgatással rendelhetünk az élekhez számokat.



10. ábra Customer Behavior Model Statechart példa

3.6 Terhelésgenerálás

A terhelésgenerálás 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 forgatókönyveknek az eloszlását is beállíthatjuk annak érdekében, hogy a virtuális felhasználók cselekedetei minél jobban illeszkedjenek a valós felhasználókéhoz.

3.6.1 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 kiszolgálót. Viszont több ezer embert bevonni a tesztelésbe nagyon költséges feladat lenne. Ehelyett számos eszköz elérhető, melynek segítségével a felhasználók tevékenységeit szimulálni tudjuk pl. Visual Studio Team System, Web Test és Load Test funkciója, a Rational Performance Tester vagy a mérésen használt JMeter (ld. 4.2. fejezet).

3.7 Teljesítményteszt tervezése

A teljesítménytesztelés három fázisból áll: tervezés, tesztfuttatás, elemzés. A tervezés során meg kell határozni a célokat, termékeket (objectives and deliverables) és az elvárásokat, definiálni kell a terhelésmodellt és a mért teljesítménymetrikákat. Kijelölni, hogy hogyan és mikor futtathatók a tesztek, milyen eszközök támogatják a tesztelést. A teszttervet létre kell hozni, majd ez alapján el kell készíteni a tesztet futtató programokat, szkripteket. Létre kell hozni egy tesztkörnyezetet, megfelelő teljesítménymonitorozással, majd ezen futtatni a teszteket és összegyűjteni az adatokat. Végül elemezni kell az eredményeket, ez alapján fejleszteni a rendszert, majd új teszteket tervezni. [5] [11]

A legfontosabb kérdések, amelyre a teljesítménytesztelés során választ kell kapnunk:

- Képes lesz-e a webalkalmazás az elvárt számú felhasználó megfelelő minőségű kiszolgálására? Milyen költséggel?
- Milyen terhelésnél kezd el degradálódni a szolgáltatás?
- Mik a degradálódás üzleti következményei?
- Hogyan lehet a webalkalmazás terhelhetőségét növelni? Milyen költséggel?
- Hogyan kell az éles rendszert monitorozni, hogy valamely alrendszer szaturációja esetén be lehessen avatkozni?

3.7.1 Tervezési fázis

A tervezési fázis legfontosabb terméke a teszterv, ez a következő elemekből áll:

- teljesítménytesztelési célok,
- terhelésmodellek,
- felhasználói modell,
- teszt procedúrák,
- a tesztelt rendszer konfigurációja,
- a mért metrikák.

További termékei a tervezési fázisnak:

- szoftver eszközök (toolok) értékelése, és a kiválasztásról szóló jelentés,
- teszt szkriptek,
- további az adott feladathoz specifikus termékek.

Terhelésmodelleket tipikusan egy CBMS vagy ahhoz hasonló felhasználói modellel és az azt kiegészítő információkkal írunk le (pl. hány felhasználó van jelen egyszerre a rendszerben, ezt akár az idő függvényében definiálva). Ha többféle szolgáltatást is nyújt a rendszer, akkor definiálni kell a terhelésmodellben, hogy ezeket egyszerre vagy külön-külön kell tesztelni, továbbá hogy egyenként milyen a terhelés.

A terhelés méretezése során először (ha még nem üzemel az éles rendszer) a specifikációra kell hagyatkozni. A következő lépés néhány felhasználó böngészésének (felhasználási adatok, usage data) rögzítése, és ennek a felskálázása a várható terhelés szerint. Végül, ha már üzemel egy éles rendszer, akkor annak a naplói, az ott mért metrikák alapján lehet tervezni a terhelést.

Az alábbi táblázat adhat iránymutatást, hogy egy adott terhelés (adott számú felhasználó) szimulálásához milyen hardverelemekre lehet szükség. Nyilván ez csupán iránymutató jellegű, erősen függ a kliens oldali feldolgozás nehézségétől és a használt terhelésgeneráló eszköztől.

Hardware Configuration	Max # Virtual Users per Agent
4xPIII, 1GHz, 4 GB RAM	5900
2xPIII, 1GHz, 2 GB RAM	3600
1xPIII, 1GHz, 1 GB RAM	1900
1xPIII, 1GHz, 512 MB RAM	800

11. ábra Felhasználók szimulálásához szükséges hardvereszközök, forrás: [6]

3.7.2 Tesztelési fázis

A tesztelési fázis termékei a következők:

- tesztfuttatási eredmények,
- nem funkcionális hibákról (pl. elégtelen teljesítmény) szóló jelentés,
- funkcionális hibákról szóló jelentés (pl. bugok).

A teljesítményteszteket a lehető leghamarabb el kell kezdeni, és gyakran ismételni. Az alkalmazásnak teljes funkcionalitással kell működnie.

A teszthez szükséges adatokat (pl. felhasználónevek, jelszavak) előre létre kell hozni vagy gondoskodni kell automatikus generálásukról. A tesztrendszernek (SUT) helyes konfigurációval készen kell állnia, kiegészítve a szükséges teljesítménymonitorozással, a definiált metrikák szerint.

Az egyes tesztfuttatásokhoz definiálni kell a felhasznált szálak számát, a futtatás időtartamát és időzítését, globális szolgáltatás esetén a felhasználók földrajzi eloszlását. Továbbá meg kell adni egyes

kérések közötti késleltetési időket (pl. valószínűségi eloszlással), a bemelegítési és lehűlési időket, az egyes metrikák elfogadott tartományait.

3.7.3 Elemzési fázis

Az elemzési fázis termékei a következők:

- A mért adatok elemzése, összehasonlítás analitikus eredményekkel,
- Rendszeres státuszjelentés,
- Végső teljesítményteszt-jelentés, javaslat az optimalizációra és új tesztekre.

Az elemzés nehézsége, hogy a teljesítménytesztelés hatalmas (akár sok GB) adatmennyiséget eredményezhet, illetve ennek megértése komoly szakértelmet igényel: mely metrikák vannak hatással egymásra, mely eredmények relevánsak, milyen hasznos információ nyerhető ki ebből. Gyakran az eredmények elemzése nehezebb feladat, mint maga a teszt végrehajtása.

Természetesen az elemzés során a legfontosabb a (lehetséges) problémaforrások, szűk keresztmetszetek azonosítása. Amint ezek megvannak, javaslatokat lehet tenni a módosítása, optimalizációra. Néhány tipikus problémára lehetséges megoldásokat mutatunk be példaként (természetesen ezen kívül rengeteg egyéb hiba is elképzelhető, és a javításuk is történhet sok más módszerrel is):

- Magas CPU kihasználtság:
    - A CPU korszerűbbre cserélése
    - További gépek rendszerhez adása terheléselosztó fürtben
    - Algoritmusok optimalizálása
- Magas válaszidő:
    - Egy nagyobb oldal több kisebb oldalra való bontása
    - Cache-ek alkalmazása
- Sok diszk IO művelet:
    - Memóriakihasználtság vizsgálata, swappelés elkerülése
    - SSD-k alkalmazása HDD-k helyett

4 A mérés szoftvereszközei

4.1 Apache Tomcat Connector

A mérésen a “Konfigurációs leírók modell-alapú fejlesztése” mérés alkalmával létrehozott konfigurációs állományokat fogjuk használni. Továbbra is szükség lesz a terheléselosztó működésének, beállításának ismeretére, ezeket a következő oldalon át lehet ismételni, amennyiben tudását bizonytalannak ítéli: http://tomcat.apache.org/connectors-doc/generic_howto/loadbalancers.html .

4.2 JMeter

A mérés során az Apache JMeter alkalmazást fogjuk használni a teljesítménytesztelésre. Ez egy Java nyelven írt, kliens-szerver architektúrájú alkalmazások tesztelésére szolgáló terhelésgenerátor. Képes a

legfontosabb szoftverelemek: web alkalmazások, FTP-, adatbázisszerverek, LDAP, JMS és egyéb rendszerek tesztelésére.

A JMeter használatának megismeréséhez az alábbi dokumentációt figyelmesen olvassák el:

- Bevezetés
  - http://jakarta.apache.org/jmeter/usermanual/get-started.html
- A webes teszteléssel kapcsolatos oldalak:
  - http://jakarta.apache.org/jmeter/usermanual/build-test-plan.html
  - http://jakarta.apache.org/jmeter/usermanual/test_plan.html
  - http://jakarta.apache.org/jmeter/usermanual/build-web-test-plan.html
- HTTP Proxy használata tesztek rögzítéséhez
  - http://jakarta.apache.org/jmeter/usermanual/jmeter_proxy_step_by_step.pdf

A következő JMeter fogalmakat ismerni kell: Test Plan, Thread Group, Element, és a legfontosabb Element csoportba tartozó eszközök.

A JMeter dokumentációja nem része a mérés elején írandó beugró anyagának, ugyanakkor aki elhanyagolja a JMeter megismerését várhatóan nem tudja befejezni a mérést.

5 Ajánlott irodalom

Ezen fejezet nem része a kötelező mérési felkészülésnek, az itt hivatkozott anyagokat a téma iránt érdeklődő hallgatók számára ajánljuk. A mérésen ezekről nem lesz szó, csak a már fent ismertetett anyagrészről.

Az alábbiakban a legmodernebb terheléselosztási, teljesítménymodellezési problémákról és megoldásaikról adunk egy rövid áttekintést néhány hivatkozáson keresztül. A cikkek teljes szövegét a http://scholar.google.com keresőn keresztül könnyen meg lehet találni.

Terheléselosztás:

M. Colajanni, Philip s. Yu, Valeria Cardellini: „Scalable Web-server Systems: Architectures, Models and Load-balancing Algorithms", Sigmetrics 2000

Karger, D. R. and Ruhl, M. 2004. Simple efficient load balancing algorithms for peer-to-peer systems. Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures 2004 http://research.google.com/pubs/archive/33339.pdf

Rao, A., Lakshminarayanan, K., Surana, S., Karp, R., and Stoica, I. Load Balancing in Structured P2P Systems. Second International Workshop on Peer-to-Peer Systems (2003).

Teljesítménymodellezés:

O. Tickoo, R. Iyer, R. Illikkal, and D. Newell, "Modeling Virtual Machine Performance: Challenges and Approaches," SIGMETRICS Performance, 2010.

D. Pariag, T. Brecht, A. Harji, P. Buhr, A. Shukla, and D.R. Cheriton, "Comparing the performance of web server architectures," ACM SIGOPS Operating Systems Review, vol. 41, 2007

Woodside, M., Franks, G., and Petriu, D. C.. "The Future of Software Performance Engineering". International Conference on Software Engineering, 2007

6 Ellenőrző kérdések

A mérés elején beugró teszt lesz, a teszt a segédlet teljes anyagára vonatkozik, nem csak az ellenőrző kérdésekben lefedett részre. Az ellenőrző kérdések megválaszolása csupán kiindulási pont a felkészülés (ön)ellenőrzésére.

1. Mit jelent a skálázhatóság informatikai rendszerekben?
2. Milyen típusú skálázhatóságot ismer?
3. Mit jelent a terheléselosztás?
4. Milyen terheléselosztási topológiákat ismer?
5. Milyen terheléselosztási algoritmusokat ismer?
6. Problémát jelentenek a sessionök egy terheléselosztással működő rendszerben?
7. Mit jelent a szűk keresztmetszet informatikai rendszerekben?
8. Mit jelentenek a következő rövidítések: SLA, SUT, CBMS?
9. Mi a különbség a load test és a stress test között?
10. Mit jelent a metrika?
11. Soroljon fel néhány ismert, a teljesítmény jellemzésére használt metrikát!
12. Mi a CBMS, és mire használják?

7 Hivatkozások

[1] J.D. Meier, C. Farre, P. Bansode, S. Barber, D. Rea (Microsoft Corporation): „patterns & practices: Performance Testing Guidance for Web Applications", http://www.codeplex.com/PerfTestingGuide

[2] Kapui Ákos: „Webalkalmazások teljesítményvizsgálata és skálázhatósága", BSc szakdolgozat, BME 2008, konzulens: Micskei Zoltán

[3] ITWare JMeter oldal, http://www.itware.eu/szakteruleteink.php?pgid=jmeter

[4] PerformanceWiki, http://performancewiki.com/home.html

[5] Department of Information Engineering and Computer Science, Feng Chia University, http://140.134.26.7/selabwiki/index.php/Performance_Testing

[6] B. Das, P. Mane „Scalability Factors for JMeter In Performance Testing Projects", STeP-IN, Performance Testing 2008, http://www.scribd.com/doc/3805411/Scalability-Factors-of-JMeter-in-Performance-Testing-projects

[7] Van Solingen, Rini; Egon Berghout (1999). The Goal/Question/Metric Method. McGraw-Hill Education. http://www.iteva.rug.nl/gqm/GQM%20Guide%20non%20printable.pdf

[8] Györfi László, Győri Sándor, Pintér Márta: Tömegkiszolgálás, Műegyetem Kiadó, 2003

[9] Randolph Nelson, „Probability, stochastic processes, and queueing theory: the mathematics of computer performance modelling”, Springer, 1995

[10] Google Let's make the web faster, http://code.google.com/speed/

[11] H. Q. Nguyen, B. Johnson, M. Hackett, R. Johnson, „Testing Applications on the Web: Test Planning for Mobile and Internet-Based Systems”, Wiley, 2003

[12] L. Williams and C. Smith, "Web application scalability: A model-based approach," *Computer Measurement Group Conference (CMG'04)*, 2004.