

M Ű E G Y E T E M 1 7 8 2

Budapesti Műszaki és Gazdaságtudományi Egyetem  
Méréstechnika és Információs Rendszerek Tanszék

## **Eclipse Rich AJAX Platform alapú webalkalmazások teljesítmény analízise**

BSc szakdolgozat

**Áshin László**

Konzulens:

**Ráth István**

PhD hallgató

Budapest, 2008. december 12.

*Szeretném megköszönni konzulensem, Ráth István munkáját, melynek során hasznos tanácsokkal látott el a dolgozattal kapcsolatban, és ötleteivel, megjegyzéseivel segítette a munkámat.*

# Nyilatkozat

Alulírott Áshin László, a Budapesti Műszaki és Gazdaságtudományi Egyetem mérnök informatika szakos hallgatója kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, és a diplomatervben csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben de átfogalmazva más forrásból átvettem, egyértelműen a forrás megadásával megjelöltem.

.....  
Áshin László

# Tartalomjegyzék

<b>1. Bevezető</b>	<b>7</b>
1.1. Az alapfogalmak ismertetése . . . . .	7
1.1.1. WEB 2.0 és AJAX . . . . .	7
1.1.2. Eclipse . . . . .	9
1.1.3. Rich Client Platform . . . . .	9
1.1.4. Rich AJAX Platform . . . . .	12
1.2. A teljesítménymérés problémaköre . . . . .	12
1.2.1. Teljesítménymérés webes környezetben . . . . .	14
1.2.2. RAP alapú alkalmazások mérése . . . . .	14
<b>2. Korábbi eredmények</b>	<b>16</b>
2.1. Webes benchmark-ok . . . . .	16
2.1.1. Böngésző forgalmát lemásoló szoftverek . . . . .	16
2.1.2. Böngészőt kiegészítő szoftverek . . . . .	18
2.2. WEB 2.0 . . . . .	20
2.3. Rich AJAX Platform . . . . .	21
<b>3. A mérési környezet</b>	<b>22</b>
3.1. A mérési környezet kifejlesztése RAP alkalmazásokhoz . . . . .	22
3.1.1. A feladat megoldása JMeter-rel . . . . .	22
3.1.2. A szerver oldali erőforrásigény mérése . . . . .	24
3.1.3. A kliens oldali erőforrásigény mérése . . . . .	24
3.2. A szerver oldali mérési összeállítás . . . . .	25
3.2.1. A méréshez használt szoftverek . . . . .	25
3.2.2. A méréshez használt hardverek . . . . .	26
3.3. Elvárt eredmények . . . . .	26
3.4. A mérés menete . . . . .	27
3.4.1. A RAP szerver oldalának mérése . . . . .	27
3.4.2. A RAP kliens oldalának mérése . . . . .	27
3.5. Mérési eredmények és azok analízise . . . . .	28
3.5.1. A szerver oldal . . . . .	28
3.5.2. A kliens oldal . . . . .	35

<b>4. Összefoglalás</b>	<b>37</b>
4.1. Elért eredmények . . . . .	37
4.2. Javítási, bővítési lehetőségek . . . . .	37

## Áttekintés

Jelen dolgozat fő témája a modern AJAX alapú webes alkalmazások teljesítmény analízise, különös tekintettel az Eclipse Rich AJAX Platform alapú megoldásokra. A tisztán szerver oldali kódokat tartalmazó környezetekkel ellentétben új kihívásokkal kerülünk szembe, ha mérnünk kell azt, hogy kiszolgálónk mekkora terhelést visel el. Az AJAX alapú alkalmazások kliens oldalon futó kódokkal és aszinkron JavaScript kommunikációval járulnak hozzá a helyzet komplexitásához, ennek fényében kell a mérési módszereinket kiegészíteni.

A bevezető rész felvázolja a probléma részleteit, majd áttekintésre kerülnek az általánosan használt módszerek. Ezután az alkalmazott mérési elrendezés lesz ismertetve. A dolgozat az eredmények ismertetésével és analízisével folytatódik. Az összefoglaló részben az elvégzett munka áttekintése olvasható, illetve a további fejlesztési lehetőségek kerülnek bemutatásra zárásképpen.

## **Abstract**

The main topic of this paper is performance analysis of state-of-art AJAX-based web applications especially of Eclipse Rich AJAX Platform ones. As opposed to purely server side environments new kinds of challenges have to be faced in case of measuring the load can be handled by the server itself. AJAX-based applications make the situation more complex by adding client side code execution and asynchronous JavaScript communication. Measurement strategies have to be updated according to these aspects.

Introduction chapter will show the details of the problem, after that generally used methods will be reviewed. Then the used measurement setup will be exposed. The paper will continue with showing measurement results and their analysis. Last chapter will review the work done and introduce some future work possibilities.

# 1. fejezet

## Bevezető

A fejezet ismerteti az alapvető problémát, valamint az ennek megértéséhez szükséges lényeges fogalmakat.

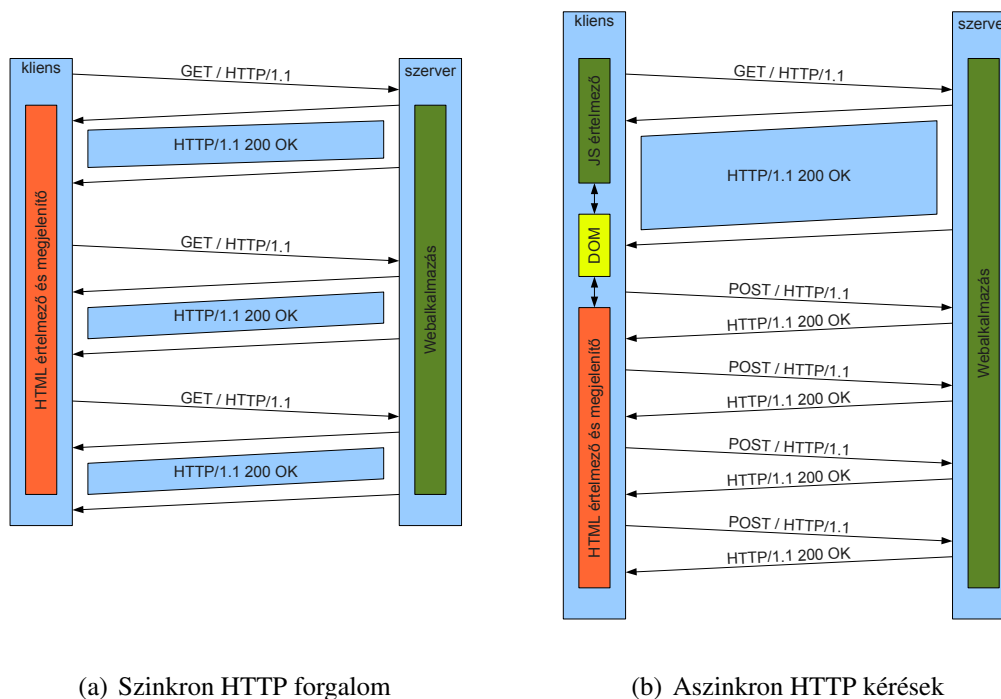
### 1.1. Az alapfogalmak ismertetése

Az alábbiakban ismertetésre kerülnek olyan fogalmak, melyek segítenek megérteni a hétköznapi webalkalmazások és a modern AJAX alapú alkalmazások működése közötti különbségeket, valamint ezek mérése közötti különbségeket. A fejezetben látni fogjuk, hogy pontosan mi is az Eclipse Rich AJAX Platform lényege.

#### 1.1.1. WEB 2.0 és AJAX

Az utóbbi időben a web fejlődése új irányvonalat vett. Internetes közösségek alakultak ki, ahol az egyének a weben keresztül elérhető szolgáltatásokon keresztül tartják egymással a kapcsolatot. Ilyen webes szolgáltatások a kapcsolati hálók, a webes enciklopédiák (a wiki oldalak), a videó megosztók, és a webes naplók (blog-ok). A WEB 2.0 az internetes együttműködés gyorsítását célozza megoldani, ami azt jelenti, hogy a szolgáltatások könnyen elérhetőek legyenek, használatuk egyszerű legyen, és gyorsan lehessen velük dolgozni. [16] Az új igények innovációkat kényszerítettek a webportálok kezelőfelületén, és így született meg az aszinkron kommunikáció bevezetésének ötlete.

Azelőtt a weboldalak követték a HTTP protokoll kifejlesztése idején kigondolt elveket, miszerint a kommunikáció szigorúan szinkron módon valósul meg az 1.1(a) ábra szerint. [11] Ez azt jelenti, hogy minden egyes felhasználói interakció az adott weblap teljes egészének ismételt betöltődését eredményezte. Az ismételt betöltődés jelentős hátránya, hogy a felhasználó várakozni kényszerül, amíg a szerver által generált tartalom elkészül és átjut a hálózaton a kliens oldalon futó böngészőbe, mely elvégzi az annak megjelenítéséhez kellő műveleteket. Ez a lépéssorozat olykor akár több másodpercesre is nyúlhat amennyiben bármely művelet hosszabb időt vesz igénybe. Egy hétköznapi alkalmazás futtatása során sem tesz jó benyomást ha egy felhasználói beavatkozás hatása nem látható pár száz milliszekundumon belül, ezzel szemben egy



1.1. ábra. Szinkron és aszinkron HTTP kommunikáció

webes alkalmazásban ekkora késleltetés könnyedén összejöhet már a hálózati átvitelből is. A teljes weblap összeállítása a szerveret terheli, átvitele az átviteli csatornát, megjelenítése pedig a kliens oldalon a böngészőt.

Sokszor azonban az egész tartalomnak csak egy kicsiny része más az előző állapothoz képest. Adódik a gondolat, hogy csak a változásokat küldje át a szerver. Ha például egy gomb megnyomásának hatására csak egy szövegmező tartalma változik (tipikus eset egy űrlap hibás kitöltése esetén), akkor kizárólag a szöveges mező új tartalmát kell a szervernek elküldenie. Ezzel a hálózati forgalom jelentős részét is megspórolhatjuk nem is beszélve arról, hogy a böngészőnek sem kell a teljes weboldalt újra értelmeznie. A kommunikációt az 1.1(b) ábra illusztrálja. A felvázolt megoldáshoz szükséges az, hogy a kliens oldal bizonyos szintig önállóan képes legyen kezelni a felhasználói interakciókat, és azokat aszinkron hívások formájában továbbítani a szerver felé. Az aszinkron hívások előnye, hogy a felhasználói felület a kérés elküldése és a válasz megérkezése között teljes mértékben használható marad, és rendelkezésre áll a felhasználó számára. Miután a szervertől kapott válasz megérkezett, ugyanezen kliens oldali kód felelős a válasz által kiváltott módosítások végrehajtásáért a web dokumentum modelljében. Ezt a megoldást képviseli az AJAX (Aszinkron JavaScript és XML) úgy, hogy a kliens oldalon futó kód JavaScript, az átviteli csatornán továbbított kérés és válasz pedig XML formában realizálódik. [9]

Mint ahogyan az az 1.1 ábrán is látható, a kétfajta működési mód közül az aszinkronnál kezdetben lehetséges, hogy egy nagyobb adathalmazt kell a kliensnek letöltenie. Ez az a JavaScript kód, mely a további kéréseket kezeli, és a felhasználói felületen a különféle kezelőelemeket

megvalósítja és működteti. Ennek letöltése után azonban már csak azok a kisméretű csomagok mozognak a hálózaton, melyek az egyes aszinkron kéréseket hordozzák. Méretük jóval kisebb a teljes oldal méreténél.

Az AJAX tehát azzal járul hozzá a WEB 2.0-hoz, hogy a felhasználói felület minőségét növeli kényelmi szempontból. Az AJAX elvén működő webalkalmazások nem tűnnek olyan szaggatottnak a felhasználók számára, sokkal inkább hasonlítanak asztali, azaz vastagkliens alkalmazásokra.

### 1.1.2. Eclipse

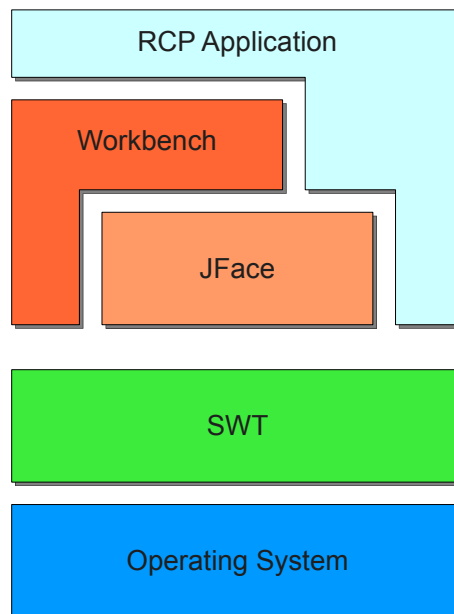
Az Eclipse egy integrált fejlesztői környezet és egy szoftver platform. [2] Kezdetben egy Java fejlesztői környezet készítése volt a projektet indító szándéka, ám kezük munkája egy bővíthető alkalmazás keretrendszerre nőtte ki magát. Mint tudjuk, a keretrendszerek általában egy absztrakciós réteg szerepét töltik be, vagyis a felettük futó alkalmazás számára különféle magas szintű szolgáltatásokat biztosítanak a könnyebb fejleszthetőség érdekében. Nincs ez másként az Eclipse esetében sem. Az Eclipse az alatta lévő Java futtatókörnyezet szolgáltatásait kibővíti egy dinamikus modul rendszerrel.

Más integrált fejlesztői környezetekben az egyes részek nincsenek ilyen élesen elválasztva, sokszor az egyes funkciókat betöltő programrészek bele vannak kódolva az alkalmazásba, és ettől az nehezebben látható át, nehezebben fejleszthető. Általában igaz, ha egy nagy feladatot kell megoldani, célszerű azt először különállóan kezelhető részekre bontani. Ez az ötlet áll az Eclipse modulrendszer mögött is. A modulrendszer felelős az egyes modulok életciklusának kezeléséért, feladata a modulok betöltése, a modulok közti kapcsolatok és függőségek menedzselése, és a már nem használt modulok eltávolítása. Az Eclipse ezt az Equinox nevű komponens felhasználásával oldja meg, ami egy futtató rendszer szerepét tölti be. Az Equinox az OSGi szolgáltatás platform implementációja, ami különféle rétegeket, programozási interfészeket és szolgáltatásokat definiál. [10] Az OSGi talán legalapvetőbb eleme az imént említett dinamikus komponens modell, amely a Java virtuális gép hiányzó elemeit pótolja.

### 1.1.3. Rich Client Platform

A Rich Client Platform (RCP) vastagkliens alkalmazások fejlesztésére alkalmas komponens alapú platform (1.2 ábra). [3] Tartalmazza a futáshoz szükséges keretrendszert, vagyis az Equinox-ot, valamint néhány komponenset. Ilyen komponensek például a megjelenítéshez használható ún. widget toolkit, azaz vezérlőelem tárház, amiből a grafikus felhasználói felület épül fel. További RCP komponenseket alkotnak a workbench elemei, ebben helyet foglalnak a különböző nézetek, perspektívák. Az Eclipse nem más, mint egy RCP alkalmazás kibővíve számos olyan komponenssel, melyek a különféle fejlesztői igényeket elégítik ki. A másik irányból nézve az RCP az a minimális Eclipse plug-in halmaz, melyekkel egy bármilyen vastagkliens alkalmazás felépíthető.

A plug-in-ek, vagyis az Eclipse alkotó komponensei vitathatatlan előnyt nyújtanak, hiszen a fejlesztő kezébe adják a komponens alapú szoftverfejlesztés lehetőségét. A komponens alapú



1.2. ábra. Rich Client Platform

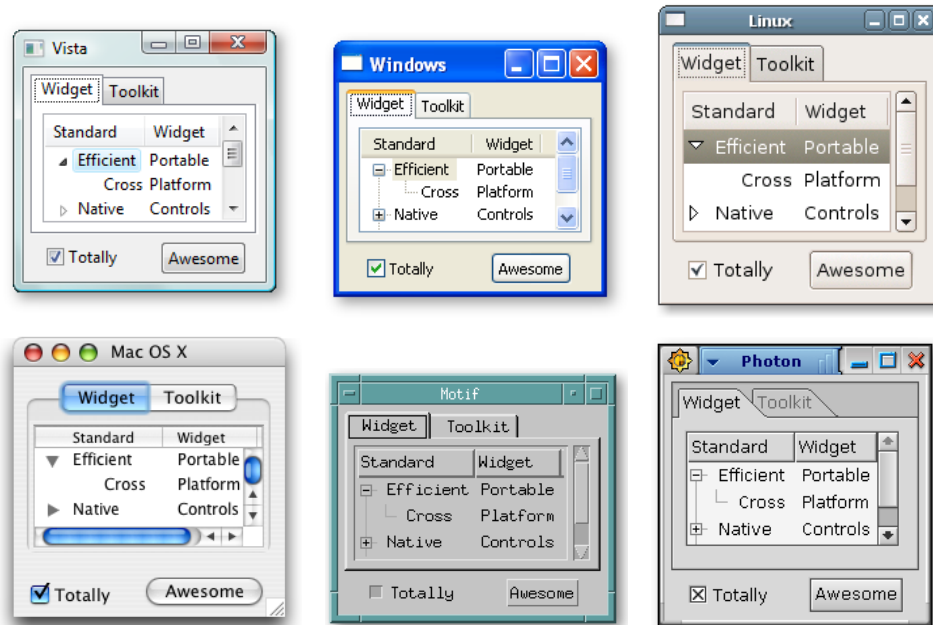
szoftverfejlesztés előnye, hogy a fejlesztőnek nem kell olyan alacsony szintű problémákkal foglalkoznia, melyek más vastagkliens alkalmazások kialakításakor gyakorta előfordulnak, hisz ezekre nagy valószínűséggel már van megoldás mások által implementált komponensek formájában. Ez nemcsak gyorsabbá teszi a fejlesztést, de elfedi az eltérő platformok egyedi jegyeit, és a fejlesztett program így platformfüggetlenné válik, tehát az RCP világban könnyebb platformfüggetlen alkalmazást fejleszteni, ami futhat különböző operációs rendszereken kihasználva maga a Java futtatókörnyezet által biztosított hordozhatóságot is.

### Standard Widget Toolkit

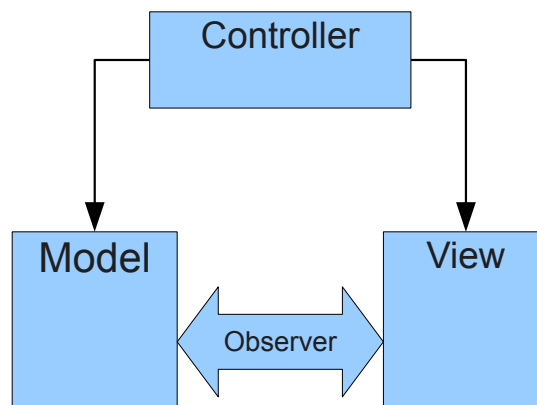
A Standard Widget Toolkit egy nyílt forrású widget könyvtár, amely az egyes felhasználói felületi elemek megjelenítéséről gondoskodik. Egységes megjelenést biztosít és programozói oldalról elrejti a különböző platformok egyedi vonásait. Az SWT alacsony szinten oldja meg a widget-ek kezelését, minden platformon az ott jelenlévő elemeket használja. (1.3 ábra) [3]

### JFace

A JFace absztrakciós szintként szolgál, ezzel kiegészítve az alatta lévő widget könyvtárat. Magasabb szintű szolgáltatásokkal bővíti ki a megjelenítést végző egységet, ilyen például a JFace jól átgondolt eseménykezelő modellje. A JFace emellett Model-View-Controller szemléletet helyez az SWT fölé.



1.3. ábra. SWT: platformspecifikus widget-ek használata



1.4. ábra. Model-View-Controller

A Model-View-Controller (MVC) architektúra a komponens alapú rendszerhez hasonlóan a felelőségek elosztását oldja meg. (1.4) Az MVC felhasználói felületek és a mögöttük álló program állapot összekapcsolását kezeli úgy, hogy a modell állapotot tároló egység minél szűkebb interfészen érintkezzen a megjelenítést végző egy vagy több nézeti egységgel. A szűk interfész abból a szempontból előnyös, hogy a szoftver belső működés módosítása közben nem kell azzal bajlódni, hogy az egyes apró változtatások hatásai milyen formában jelentkeznek majd a felhasználói interfészekben. Arra elég akkor figyelni, amikor a nézeti struktúra fejlesztése zajlik. A modell és a nézeti egységek egymás közti kommunikációjára az MVC architektúrában általában az Observer tervezési mintát alkalmazzák. Ez úgy működik, hogy az egyes nézetek feliratkoznak a modell állapotváltozásaira. A feliratkozott nézeteket a modell a kiválasztott esemény hatására értesíti, így azok lekérdezhetik annak belső állapotát, és frissíthetik a felhasználói felületet megfelelően.

### 1.1.4. Rich AJAX Platform

Az előzőekben láthattuk az AJAX alapú webalkalmazások és a Rich Client Platform alapú vastagkliens alkalmazások előnyeit. Nyilvánvaló, hogy az AJAX a WEB 2.0 elterjedésével előtérbe kerül, a komponens alapú szoftverfejlesztés pedig elengedhetetlen a gyors fejlesztéshez. Ha ezt a kettőt sikerülne összegyúrni, akkor egy ütőképes megoldást kapnánk a web új kihívásaira. Így született meg a Rich AJAX Platform (RAP). [13] Akár a neve, architektúrája is hasonló a RCP-hez, ahogy azt az 1.5 ábra is mutatja.

Látható, hogy az RCP és a RAP a JFace-től felfelé megegyezik. A legfelső rétegben futó alkalmazások ettől lefelé csak az SWT, illetve RWT nevű rétegeket látják, melyek interfésze nagyon hasonló. Ez fontos, hiszen azt jelenti, hogy egy RCP alkalmazás elméletileg könnyen átültethető RAP-ra. Ezzel egy meglévő vastagkliens alkalmazást alakíthatunk át AJAX-os vékonyklienssé anélkül, hogy foglalkoznunk kellene például azzal a problémával, melyik böngészőben is fog futni programunk.

#### RWT és qooxdoo

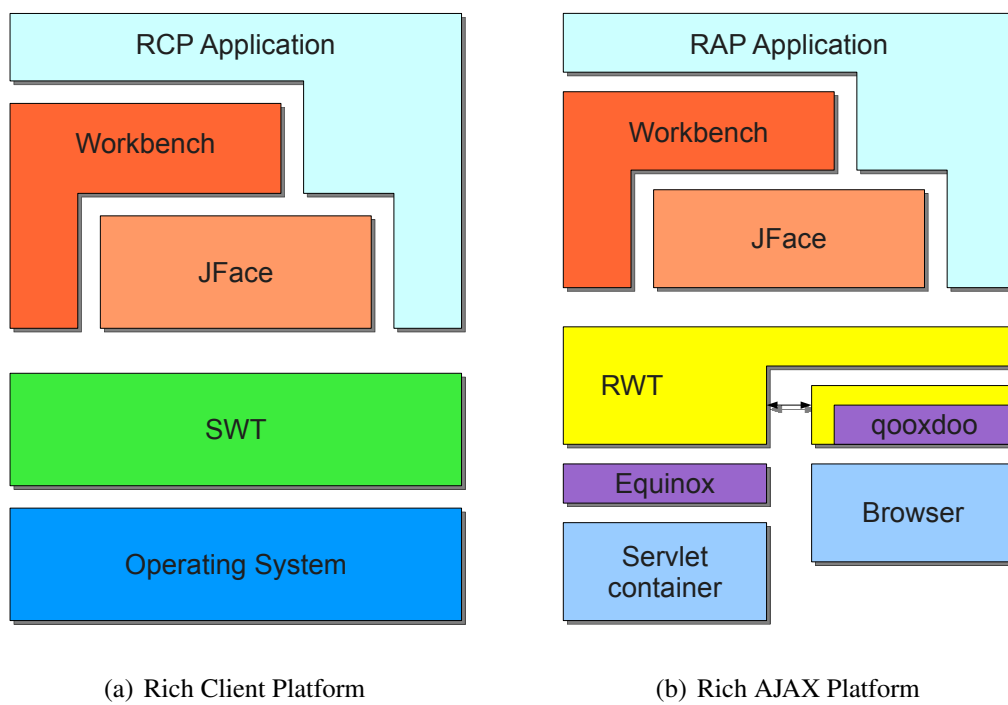
A RAP Widget Toolkit (RWT) egy SWT-hez hasonló megoldás. Feladata, hogy összekösse a szerver vastagkliensnek megfelelő interfészét a kliensben futó JavaScript keretrendszerrel. A kliens oldalon megjelenő widget-eket a qooxdoo nevű AJAX keretrendszer biztosítja.

#### A szerver oldal

A RAP alkalmazás szerver oldalán egy webszerver áll, melyen belül egy Java alkalmazás szerver fut. Az ebbe telepített Equinox futtatja az alkalmazás betöltött komponenseit.

## 1.2. A teljesítménymérés problémaköre

Egy szoftver teljesítményének ismerete fontos az azt kiszolgáló rendszer tervezésekor, hiszen annak elemeit méretezni kell. A méretezést a várható igényeknek megfelelően kell elvégezni



(a) Rich Client Platform

(b) Rich AJAX Platform

1.5. ábra. RCP és RAP összehasonlítása

bizonyos mértékű ráhagyással, hogy a rendszer biztosan megfeleljen bármilyen körülmények között a vele szemben támasztott követelményeknek. A rendszer teljesítményét megtudhatjuk analitikus módon számításokkal. Ez a módszer bonyolult lehet, ráadásul nem is tudunk minden körülményt számításba venni. A teljesítmény meghatározásnak másik módja a mérés. Méréskor a mérendő rendszerre olyan bemeneti adatokat adunk, melyek közelítik a valós körülmények között kapott adatokat.

Az adatok feldolgozása közben a szoftver erőforrásokat vesz igénybe. Ilyen erőforrások például a processzor, a memória és a hálózat. Fontos metrika a feladat végrehajtásához szükséges idő, valamint az egységnyi idő alatt feldolgozott adatmennyiség. A teljesítménymérés során ezeket a paramétereket kell mérni, ezek az adott rendszer összeállítás jellemzői.

### 1.2.1. Teljesítménymérés webes környezetben

Webes környezetben a mért szoftver a szerveren futó program, mely a kliens számára összeállítja a HTML tartalmat. A tartalom összeállításához általában szükséges egy vagy több adatbázis művelet végrehajtása, ezért a webszerver mellett az adatbázis kiszolgálót is célszerű belevenni a mérésbe. A mérés során HTTP kliensek segítségével szimuláljuk egyes oldalak letöltését. Egy ilyen oldal letöltésekor a szervernek el kell végeznie az oldal összeállításához kellő összes műveletet, így erőforrásokat fog igénybe venni, és ez mérhető. Mérhető a processzorhasználat, az igénybe vett memória mérete, az átviteli hálózat kihasználtsága. A HTTP kliens pedig mérheti a kérések elküldése és az azokra érkező válaszok megkapása között eltelt időt, és az adott idő alatt kapott sikeres válaszok számát, melyek a webes rendszer legfontosabb paraméterei.

Webes rendszer esetén a késleltetés minimális értéken tartásának fontossága nem kérdéses. Gondoljunk csak a felhasználókra, akik valamilyen szolgáltatást kívánnak igénybe venni. Valószínűleg azt a céget fogják választani a lehetőségek közül, mely a legjobb benyomást teszi rájuk. A jó benyomás márpedig a weboldal megfelelő kinézete mellett a gyors kiszolgálással érhető el. Ha az ügyfélnek nem kell minden kattintás után másodperceket várni arra, hogy tovább lépessen az oldalon, akkor valószínűleg ez pozitívan fog hatni rá. Erről szól a hét másodperces szabály, mely hét másodpercnél húzza meg az elfogadható kiszolgálási szint mértékét.

A rövid válaszidő nem elégséges feltétele a jó kiszolgálásnak. Ha a weboldalon szerepelnek nagyméretű statikus tartalmak, akkor fontos, hogy azok a leghamarabb átjussanak a böngészőbe, minél hamarabb láthassa a nagyméretű képeket a felhasználó, vagy hamar elinduljon a videó lejátszás. A nagy fájlok átviteléhez nagy sávszélesség szükséges.

A webkiszolgáló feladata a kliensek kiszolgálása. Ahogy azonban a kliensek száma növekszik, nőni fog a késleltetés, és lecsökken az egyes felhasználókra jutó hálózati kapacitás. A rendszer fontos jellemzője tehát az is, hogy hány felhasználót tud ellátni úgy, hogy a szolgáltatási minőség elfogadható szinten marad.

### 1.2.2. RAP alapú alkalmazások mérése

A Rich AJAX Platform esetében a mért szoftver nemcsak a szerver oldalon helyezkedik el, hiszen AJAX-ról lévén szó a kliens oldal futtatja a szervertől kapott JavaScript kódokat. Az AJAX-os webalkalmazások mérése általában körülményesebb, hiszen a terhelés megoszlik a szerver

és a kliens oldal között, és ezt a RAP framework oldja meg, melynek belső állapotát nehéz megállapítani. Éppen ezért célszerű a processzorhasználatot és a memóriaigényt a kliens oldalon is mérni, hiszen a webes alkalmazás ezekre a paraméterekre is befolyással van.

Tovább nehezíti a helyzetet, hogy nem elég egyszeri HTTP kéréseket küldeni, hiszen AJAX környezetben egy művelet végrehajtásához több lépés, azaz több kérés-válasz üzenetváltás szükséges. Ezért egy teljes üzenetváltás szekvenciát kell küldeni a szervernek, és ez kizárja az egyszerű webes benchmark programok használatát. Ráadásul az üzenetváltás során kliens oldalon tárolt állapot (cookie-k) megőrzéséről is gondoskodni kell.

Nehézség még, hogy a szakirodalom nem bővelkedik a RAP alapú, illetve AJAX-os webalkalmazások teljesítményméréséről szóló anyagokban, így önállóan kellett felkutatnom a felhasználható módszereket és eszközöket. A következő részben ismertetem a kutatómunka során talált megoldási alternatívákat azok előnyeivel és hátrányaival.

## 2. fejezet

# Korábbi eredmények

A fejezet áttekinti a webes teljesítménymérés jelenlegi eszközeit és az AJAX alapú webalkalmazások esetében alkalmazott elveket.

### 2.1. Webes benchmark-ok

A korábban leírt módon a webes teljesítményméréshez valamilyen formában HTTP kliensek szimulációja szükséges. A kliens valós terhelés esetén persze valamilyen webböngésző, ám ha egy gépről akarunk nagy terhelést küldeni a szerverre, akkor nem biztos hogy böngészők sokaságával célszerű operálni. Ez azért nem lenne előnyös, mert minden egyes böngésző példány lefoglal bizonyos méretű memóriát, és a különböző példányok nem feltétlenül tudnak egymásról, hogy egy célra közös területeket foglalhassanak és a közös területeket megosztva használhassák. Amellett, hogy ez a megoldás pazarlóan bánik a memóriával, a mérés vezérlése nem oldható meg könnyen, hiszen gyakorlatilag egy kattintás sorozatot kellene lejátszani a böngészőkben, majd a szekvencia végén törölni a session állapotot. Erre létezik ugyan megoldás, de a célhoz sokkal rövidebb úton is eljuthatunk.

Léteznek ugyanis olyan szoftverek, melyek kifejezetten a méréshez szükséges HTTP kérések szintjén oldják meg a feladatot, tehát olyan kéréseket állítanak elő és küldenek el a webszervernek, mint amelyet egy böngésző küldene. Előnyük a sokkal kisebb erőforrásigény, ami elérhetővé teszi nagy számú felhasználó egyidejű szimulációját. A következő részben ilyen eszközök ismertetése következik.

#### 2.1.1. Böngésző forgalmát lemásoló szoftverek

##### ApacheBench

Az ApacheBench egy kis segédprogram, mely az Apache nevű webszerver mellé jár annak gyors teszteléséhez. [1] Egy parancssori interfésszel ellátott programról van szó, mellyel ellenőrizhető a telepített Apache szerver működése. A program alkalmas az egy másodperc alatt végrehajtott kérések számát mérni. Az eszköz nem igazán alkalmas univerzális mérések végrehajtására, csak

a fent leírt mérés végrehajtására. Nem tartalmazza a HTTP/1.1 protokoll minden képességének implementációját, tudása csupán a legszükségesebb dolgokra korlátozódik.

### **httperf**

A httperf szintén webszerverek teljesítményének mérésére szolgáló eszköz. [17] Az ApacheBench-hez képest széleskörűbben támogatja a HTTP protokollt. Hasonlóság az ApacheBench programmal, hogy mindkettő C nyelven került megvalósításra, így jobb memória és processzor kihasználtságot lehet velük elérni, mint a később bemutatásra kerülő Java alapú megoldások esetében.

A httperf különféle terhelés generátorokat (workload generator) alkalmaz, ezzel modulárisra téve a kérések összeállítását végző egységet. Létezik olyan terhelés generátor, amely egy definiált URI tartomány jár be, olyan amely URI-k bizonyos sorozatát kéri le, illetve olyan is, mely egy munkamenetbe szervezi a kéréseket.

Esetünkben nagyon fontos hiányossága az eddig felsorolt programoknak, hogy nem kezelnek kérés szekvenciákat, melyek egy AJAX lépéssorozat végrehajtásához elengedhetetlenek. A következőkben olyan eszközök kerülnek sorra, melyek ilyen lépéssorozatokat is tudnak futtatni.

### **Apache JMeter**

Az Apache JMeter az előzőeknél kifinomultabb megoldást ad a webszerverek teljesítményének mérésére. [7] Grafikus kezelőfelülettel ellátott Java desktop alkalmazásról van szó, amely konkrét lépéssorozatok, illetve munkamenetek futtatására alkalmas. Moduláris felépítésű, szofisztikált teszt esetek összeállíthatók vele. Kezelőfelülete első ránézésre talán kissé bonyolultnak tűnik, de az alapfunkciók megismerése után jól használható. Tartalmaz egy beépített proxy szerveret, mellyel az egyes műveletsorozatok felvehetőek, ezeket a teszt futtatása során képes visszajátsszani. Maga a teszt visszajátsszása természetesen történhet több szálon is, így nagy mennyiségű kérés indítható, ezzel mérhető a webszerver teljesítménye. A lépések végrehajtása időzíthető, véletlen jelleget adva az időzítésnek közelíthető a valódi felhasználók által előidézett terhelés. A mérési eredmények menet közben megjelennek a felületen, majd a tesztek lefutása után fájlba menthetők, vagy akár grafikusán ábrázolhatók. Egy teszt lefutása utáni állapotot mutat a 2.1 ábra.

### **Grinder**

A JMeter-hez hasonlóan a Grinder is egy Java alkalmazás. [4] Több komponensből áll, ezek a konzol, az ágens, valamint a proxy. A klienseket az ágens komponensek szimulálják, ezeket a konzolról lehet irányítani. (2.2(a) ábra) A JMeter-től eltérően itt külön programokként vannak megvalósítva a felsorolt komponensek, ami például lehetővé teszi azt, hogy különböző gépeken futtassunk ágens példányokat, így elosztottan terhelhető a webszerver. Minden ágens egyenként többszálú. A proxy komponens, melyet a 2.2(b) ábrán láthatunk, a lépéssorozatok felvételére alkalmas. A Grinder saját jython alapú script interfésszel rendelkezik, ami alkalmas a tesztek lebonyolítására, vagyis a teszt egyes lépései a scriptben metódushívások formájában valósulnak

The screenshot shows the Apache JMeter 'Aggregate Report' window. On the left, a tree view shows a 'Test Plan' containing a 'Thread Group' with several 'HTTP Request Defaults' and 'HTTP Cookie Manager' components, followed by multiple instances of a test script labeled '/raptest01/rap?nocach'. The main area displays the 'Aggregate Report' for the selected test script. The report includes a table with the following data:

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput
/raptest01/r...	2	114	135	135	93	135	0.00%	2.2/min
/raptest01/r...	2	59	75	75	43	75	0.00%	2.2/min
/raptest01/r...	2	10	12	12	9	12	0.00%	2.2/min
/raptest01/r...	2	10	13	13	7	13	0.00%	2.2/min
/raptest01/r...	2	7	8	8	7	8	0.00%	2.2/min
/raptest01/r...	2	7	8	8	7	8	0.00%	2.2/min
/raptest01/r...	2	10	14	14	6	14	0.00%	2.2/min
/raptest01/r...	2	21	36	36	6	36	0.00%	2.2/min
/raptest01/r...	2	17	21	21	13	21	0.00%	2.2/min
/raptest01/r...	2	9	10	10	9	10	0.00%	2.2/min
TOTAL	20	26	12	93	6	135	0.00%	22.0/min

2.1. ábra. Apache JMeter

meg. A proxy ilyen jython script formájában hozza létre a kimenetét, amiben a proxy-n átment kérések kerülnek felsorolásra. Ezek a kérések alapesetben nincsenek szekvenciába szervezve, hanem egymással párhuzamosan futnak le, ha egy ilyen rögzített jython scriptet a Grinder-rel futtatunk. Ezen a script testre szabásával módosíthatunk.

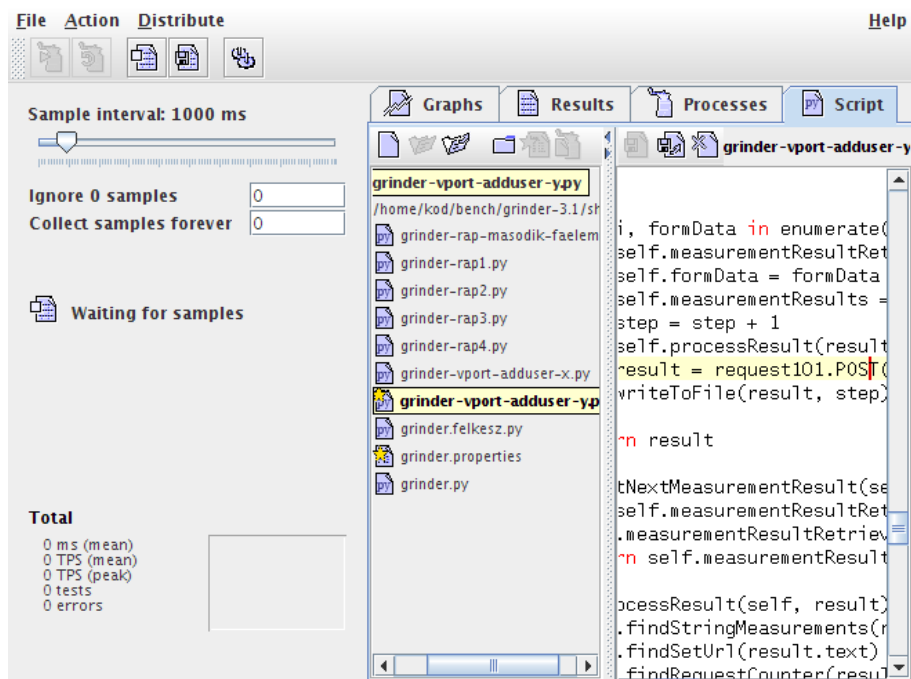
### 2.1.2. Böngészőt kiegészítő szoftverek

Az itt említésre kerülő megoldások közös pontja, hogy egy webböngészőt tesznek teljesítmény-mérésre alkalmassá. Mindegyik ebben a részben ismertetett program a Mozilla Firefox böngészőn alapul. Ennél a böngészőnél ugyanaz az alapelv figyelhető meg, mint amit az Eclipse képvisel, nevezetesen a modularitás. A Firefox esetén a különálló, utólag hozzáadható komponenseket kiterjesztéseknek nevezzük. Rengeteg ilyen kiterjesztés érhető el, ez is jelzi a böngésző népszerűségét a fejlesztők körében. Ezek a kiterjesztések sok mindenre alkalmassá tehetik a böngészőt, ebben a részben azonban csak a webes teljesítmény-mérés szempontjából érdekesek közül néhányat vizsgálunk.

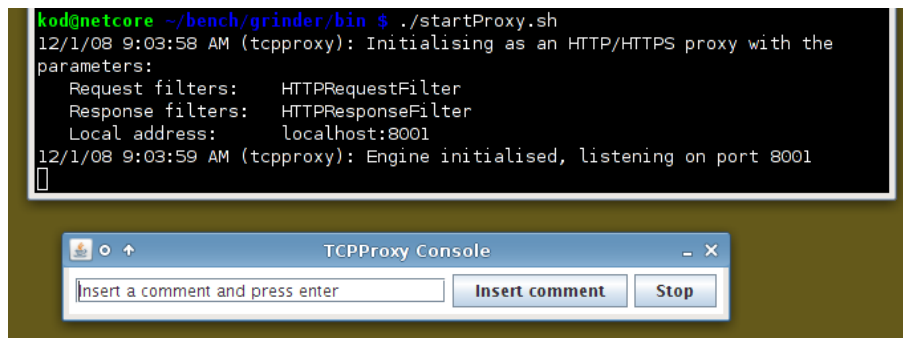
#### IMacros

Az IMacros egy olyan kiterjesztés, melynek elsődleges célja automatizálni egyes feladatokat a böngészőn belül. [5] Ismétlődő feladatok rögzíthetők vele makrók formájában, majd ezek utána visszajátszhatók. Ilyen feladat például webhelyek meglátogatása, űrlapok kitöltése. Használható webes alkalmazások tesztelésére, mivel a rögzített teszt esetek könnyen végrehajthatók vele.

Az IMacros a magas szintű felhasználói interakciókat rögzíti, nem pedig a hálózati forgalmat. A webes teszt illetve benchmark szoftverek ilyen szempontból is csoportosíthatók. A böngészőben futó megoldásoknál természetesen egyszerűbben megoldható feladat az ilyen magas szintű interakciók reprodukálása, mint a korábban tárgyalt, nem böngészőben futó alkalmazásoknál.



(a) Grinder console



(b) Grinder proxy

## 2.2. ábra. Grinder

A kiterjesztés hátránya, hogy jelenlegi változata nem támogatja a felvett lépéssorozat tömeges visszajátszását, ezért teljesítmény tesztek végzésére nem vállalkoztam vele.

## Selenium

A Selenium egy webes rendszerek tesztelésére készített eszközkészlet. [12] Alkalmas számos programozási nyelven definiált teszt futtatására a legelterjedtebb böngészők segítségével. A Selenium a Grinder-hez hasonlóan különálló komponensekből áll. Ezek a Selenium IDE, és a Selenium RC.

A Selenium IDE (integrált fejlesztőkörnyezet) egy olyan Mozilla Firefox kiterjesztés, mellyel tesztek vehetők fel, és a már rögzített tesztek szerkeszthetők.

A Selenium RC (távírányító) egy olyan interfésszel egészíti ki a támogatott böngészőket, melyen keresztül azok távolról vezérelhetők.

A Selenium nemcsak egér kattintások szintjén képes rögzíteni a felhasználói eseményeket, hanem az egér kattintások pozíciói alapján meg tudja találni a kattintás által aktivált weboldal elemet, vagyis a DOM fában keresi meg a célpontot. Ez abban az esetben hasznos, ha a tesztelt alkalmazás nem minden esetben helyezi ugyanazon pozícióra a felhasználó felületi elemeket, hisz ekkor a tesztek végrehajtásakor a rögzített koordináták nem lennének mérvadóak. A dokumentum elemeinek direkt elérésével azonban ilyen körülmények között is eredményes lehet a tesztek futtatása.

## 2.2. WEB 2.0

A WEB 2.0-ás alkalmazásokban az AJAX miatt a felsorolt programok közül bizonyos a minimális követelményeket ki nem elégítőket nem használhatunk teljesítménymérésére. A legfontosabb ilyen követelmény a kérések szekvenciáinak végrehajtása. Ennek fontossága az AJAX működési elvéből következik. Egy WEB 2.0 környezetben egyetlen magas szintű művelet végrehajtásához esetleg több tíz kérés fut le a háttérben anélkül, hogy a felhasználó mindebből bármit is észrevenne. Ezeket a lépéseket úgy kell szimulálni, hogy a webalkalmazás ne tudjon különbséget tenni egy természetes, böngészőt használó látogató, és a tesztet végző program között.

Ide tartozik a munkamenet kezelésének problémája is. Amennyiben a tesztelést végző program például nem egy munkamenetbe tartozóként kezeli a lépéssorozat kéréseit, akkor a webszerver szempontjából a kapott kérések az első kivételével teljesen értelmetlenek lesznek, ami azt jelenti, hogy a szimulált magas szintű művelet nagy valószínűséggel hatástalan lesz. Ez kihat a rendszer terhelési paramétereire is, hiszen általában a lépéssorozatok eredménye egy viszonylag intenzív adatbázis művelet a sorozat vége felé. Ha ez nem fut le, akkor éppen a nagy terhelést jelentő komponens marad ki a mérésből. Tehát a mérés során valamilyen módon visszajelzést kell kapni a tesztelt lépéssorozatok eredményéről, hogy biztosak lehessünk a kapott eredmények korrektségében.

Az eredmények korrektségét ebből a szempontból többféle módon vizsgálhatjuk. Egy lehetséges mód a kérésekre érkező válaszok vizsgálata. Ez úgy történhet, hogy az elküldött kérések mellett a teszt sorozatban a kérésekre kapott válaszokat is rögzítjük, és ha a teljesítménymé-

rés során ezektől eltérő válaszokat kapunk, akkor azt jelezzük. A módszer különleges igényeket támaszt a tesztelést végző alkalmazással szemben.

Az eredmények helyességének vizsgálata indirekt módon is végezhető. A probléma a lépéssorozat inkorrekt végrehajtása esetén a sorozat végén elhelyezkedő intenzív adatbázis művelet elmaradása volt. Ha ennek az adatbázis műveletnek az eredményességét vizsgáljuk, akkor ez jó eséllyel magának a lépéssorozat lefutási sikerességével megegyező eredményt kell kapnunk, hiszen ez azt jelenti, hogy gyakorlatilag a művelet lényegi része sikeresen végrehajtásra került a szerver oldalán.

### **2.3. Rich AJAX Platform**

Minden az előző, AJAX-ról szóló részben leírt állítás vonatkozik a Rich AJAX Platform alapú alkalmazásokra is, hiszen ez is ugyan arra az alapra épül. Ezen felül tekintettel kell lenni a keretrendszerben jelenlévő apróságnak tűnő elemekre is, melyek döntően befolyásolhatják a tesztelés sikerességét. Ilyen apróságok rejlenek a HTTP rétegen átkerülő POST típusú kérések paramétereinek kezelésében, melyeket egy jól átgondolt teljesítmény mérés előtt mindenképpen figyelembe kell venni.

## 3. fejezet

# A mérési környezet

Ez a fejezet ismerteti az általam használt mérési eszközöket, a mérések menetét, a leszűrt tanulságokat és a mért eredményeket.

### 3.1. A mérési környezet kifejlesztése RAP alkalmazásokhoz

A fejezet első részében a Rich AJAX Platform alapú alkalmazások mérését lehetővé tevő környezet kialakításáról lesz szó.

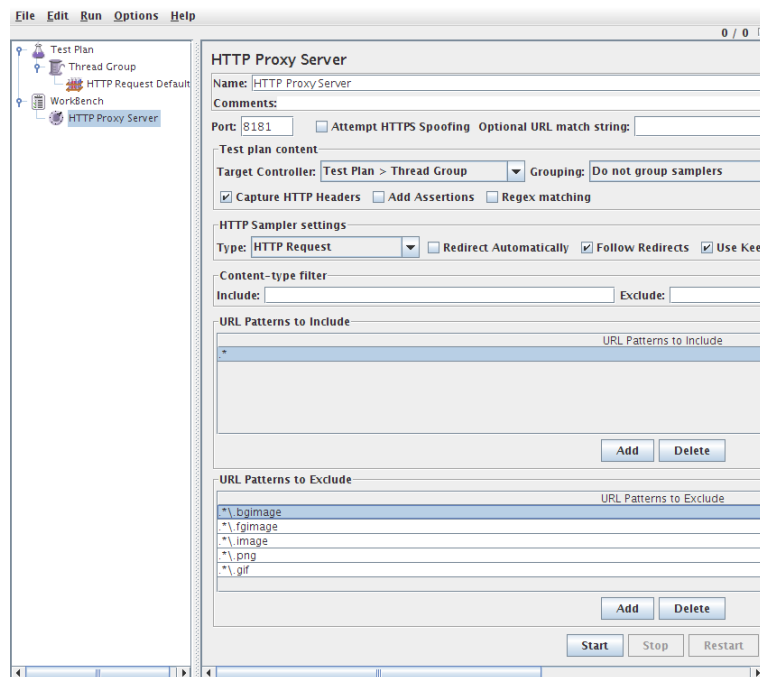
#### 3.1.1. A feladat megoldása JMeter-rel

A felsorolt, tesztelésre szolgáló alkalmazások közül a JMeter alkalmasnak tűnt a feladat megoldására. A RAP wiki oldalain is a JMeter-t ajánlották tesztelési célokra a platformhoz. [8] A böngészőben futó tesztek rágadásul el akartam kerülni, mert azok túlságosan leterhelték volna a kliens oldalt. A JMeter használata kézenfekvőnek tűnt, így megpróbáltam egy munkamenetet végrehajtani vele egy általam összeállított próba RAP alkalmazással.

A kis RAP alkalmazás néhány gombot jelenített meg, melyeket jó sorrendben megnyomva egy adatbázis rekord jön létre. Az alkalmazás tehát már adatbázis műveleteket is végrehajtott, ami egyrészt fontos a valós viszonyok modellezése szempontjából, másrészt a teszt sikeres lefutásáról is információt ad, ha az adattáblabeli módosítás jelen van a teszt után.

#### Teszt adatsor rögzítése

Első lépésben a JMeter HTTP Proxy Server komponensével felvettem a munkamenetet a későbbi visszajátszáshoz. Mivel saját alkalmazást teszteltem, először el kellett indítani az Eclipse fejlesztőkörnyezetben az alkalmazást. Meggyőződtem róla, hogy az alkalmazás fut, és eléri az adatbázist. Fontos a munkamenet változóinak törlése a böngészőben, hiszen ennek elmulasztása esetén nem indulna új session a lépéssorozat rögzítésének kezdetekor. A proxy indítása előtt be célszerű beállítani olyan URL szűrőket, melyek kiszűrik a felesleges kéréseket, esetünkben a statikus képeket. Én a 3.1 ábrán látható beállításokat alkalmaztam.



3.1. ábra. JMeter HTTP Proxy Server

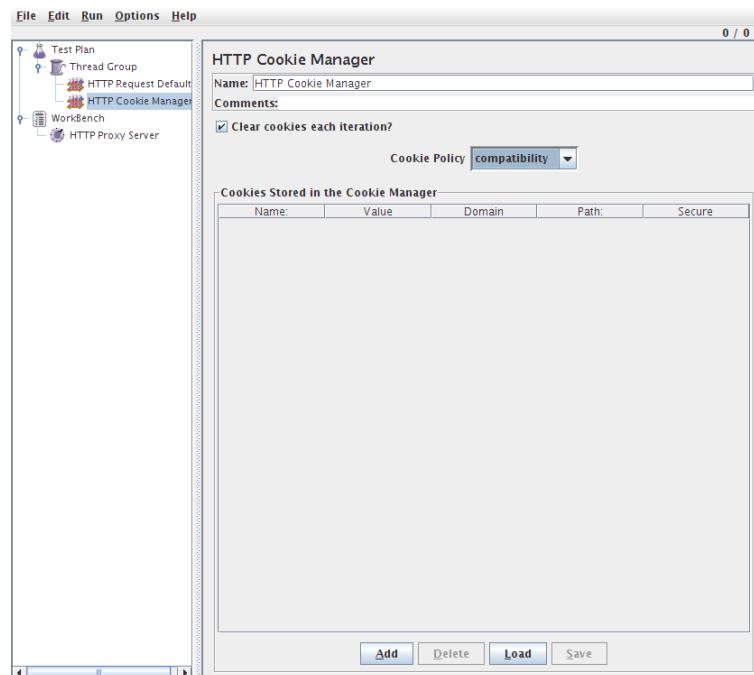
A böngészőben a proxy szerver megadása után végrehajtottam a tervezett lépéssorozatot, majd leállítottam a rögzítést. A proxy szerveren áthaladó ki nem szűrt kérések a 3.1 ábra bal oldalán látható fa nézet Thread Group nevű eleme alatt jelentek meg.

### Teszt adatsor visszajátszása

A Rich AJAX Platform alapú webalkalmazások sikeres teszteléséhez feltétlenül szükséges a munkamenet állapotváltozóinak megőrzése a közös munkamenetbe tartozó kérések között, és ennek betartásához JMeter környezetben eszköze a HTTP Cookie Manager komponens. Ezért szükséges a Thread Group elemhez hozzáadni ezt a komponenszt a 3.2 ábrán látható beállításokkal. A munkamenetek között a tároló törlődni fog, és ez a minden lépéssorozat végén új session elkezdését eredményezi.

A szálcsoporthoz továbbá beállítható az, hogy hányszor fusson le a lépéssorozat, valamint hány párhuzamos, felhasználót szimuláló szál fusson egymás mellett a teszt folyamán. Mivel egyelőre csak a teszt lefutásának sikeressége érdekelt, mindkét paramétert egyre állítottam, és bekapcsoltam a szerver által küldött válaszok naplózását. A teszt futtatásakor elégedetten tapasztaltam, hogy a rögzített válaszok alapján a munkamenet sikeresen lefutott. Ezt az adatbázis megváltozott tartalma megerősítette.

Apache JMeter-rel tehát Rich AJAX Platform alapú webalkalmazások tesztelése lehetséges.



3.2. ábra. JMeter HTTP Cookie Manager

### 3.1.2. A szerver oldali erőforrásigény mérése

A tesztelés során a szerver gépen futó alkalmazás szerver és az adatbázis szerver erőforrásigényének alakulását mérni kell. Célszerűen a processzorhasználatot, és a memóriaitigényt vizsgáltam a tesztek során a Linux kernel BSD Process Accounting szolgáltatásának felhasználásával. [15] Ezzel a módszerrel lehetőség van az egyes folyamatok futó állapotban eltöltött idejének mérésére, illetve a futásuk során összesen lefoglalt memória méretének jegyzésére. A mérés után a mért adatok az egyes folyamatok neveire vetítve jelennek meg. A mérési eredményekben az alkalmazás szerver erőforrásigényét a java, míg az adatbázis szerver erőforrásigényét a mysqlld nevű folyamatok reprezentálják.

### 3.1.3. A kliens oldali erőforrásigény mérése

AJAX-ról lévén szó a webalkalmazás futása nemcsak a HTML tartalom szerveren történő felépítéséről szól, hanem a kliens oldalon futó JavaScript kódok is végeznek munkát a dokumentum helyi manipulálásával, esetleg teljes összeállításával, és a teljes felhasználói interfész kezelésével. Ezt mérlegelve szóba kerül az erőforrásigény mérése a szerver mellett a kliens oldalán is.

### Rich AJAX Platform alapú alkalmazások esete

A kliens oldali erőforrásigények méréséhez néhány Linuxon elérhető böngészőn végeztem méréseket. A mérésekhez a szerver oldali mérés során is felhasznált BSD Process Accounting szol-

gáztatást használtam.

A kliens oldali terhelés mérése a szerver oldali méréstől természetesen elkülönítve történt, hiszen a szerver oldal mérésekor a kliens oldalon a JMeter állt.

### Általános JavaScript végrehajtás

Mint tudjuk, az AJAX technológia a JavaScript-en alapszik, vagyis a JavaScript kódok végrehajtása döntő szerepet játszik a kliens oldalon. Létezik egy SunSpider nevű JavaScript benchmark program, mely néhány előre összeállított, tipikus feladatokat ellátó kódrészlet végrehajtási sebességét méri. [14] Bár ez nem kapcsolódik szorosan a Rich AJAX platform alapú alkalmazásokhoz, segít elhelyezni a különböző böngészőkben alkalmazott JavaScript motorokat gyorsaság alapján. Az eredményeken látni fogjuk, hogy ez a mérés rámutat a JavaScript végrehajtókban alkalmazott új technológia, a JIT<sup>1</sup> [6] előnyére a konvencionális megoldásokhoz képest.

## 3.2. A szerver oldali mérési összeállítás

A szerver oldali teljesítménymérés összeállítását 3.3 ábra mutatja. Fontos, hogy a szerver és a kliens két különböző gépen fusson, hiszen ellenkező esetben a kliens erőforrás használatát nem lehetne elkülöníteni a szerverétől.

### 3.2.1. A méréshez használt szoftverek

**Eclipse Ganymede 3.4.1** a tesztelt RAP alkalmazások fejlesztéséhez, és azok WAR archívumba történő exportálásához.

**Rich AJAX Platform 1.1.1** a használt alkalmazás platform. Elérhető az Eclipse fejlesztőkörnyezethez plug-in formájában.

**Glassfish 9.1\_02 (build b04-fcs)** alkalmazás szerver.

**MySQL 5.0.70** adatbázis szerver az alkalmazások perzisztenciájának biztosítására.

**Linux 2.6.27** a gépeken használt Linux rendszermag.

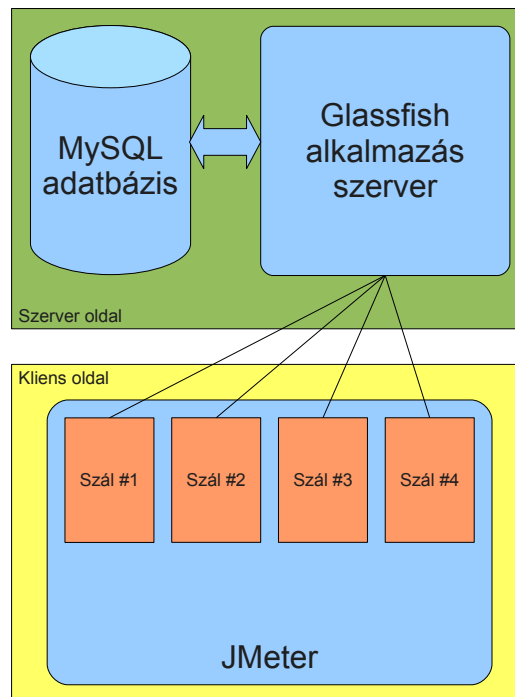
**Jakarta JMeter 2.3.2** a kliens gépen futó tesztelést végző alkalmazás.

**GNU system accounting utilities** a gépek erőforrásigényeinek monitorozását végző eszközkészlet.

**Mozilla Firefox 3.1b1** böngésző a teszt esetek rögzítéséhez.

---

<sup>1</sup>JIT: Just-in-time compilation



3.3. ábra. A szerver oldali mérés elrendezése

### 3.2.2. A méréshez használt hardverek

#### A szerver

Intel Core 2 Quad Q6600 CPU 2.4GHz, 2x4MB L2 Cache

4GB DDR2 memória 800MHz dual channel

#### A kliens

AMD Athlon XP 2400+ CPU 2GHz, 512kB L2 Cache

768MB DDR memória 333MHz

## 3.3. Elvárt eredmények

A mérés elvégzése előtt számítani lehet néhány eredményre. Ha ezek az elvárt eredmények a mért értékektől számottevő eltérést mutatnak, az mindenképp átgondolásra ad okot. A szerver oldali méréseknél az elvárások a következők:

**Késleltetés** A késleltetés azt mondja meg, hogy egy adott kérésnél a felhasználónak mennyi ideig kell várnia, míg a választ megkapja. A késleltetés várhatóan úgy alakul a mérés során, hogy több párhuzamos felhasználó esetén nagyobb értéket kapunk, vagyis minél többen terhelik a szervert, az annál lassabban képes kiszolgálni az egyes klienseket.

**Áteresztőképesség** Az áteresztőképesség a másodpercenként végrehajtott kérések számát reprezentálja. Előreláthatólag az áteresztőképesség fordított arányban fog állni a késleltetéssel, illetve a terhelő felhasználók számával, vagyis minél többen böngészik az adott tartalmat, annál kisebb lesz az időegységre jutó teljesített kérések száma.

**Adatátviteli sebesség** Az adatátviteli sebesség a szerver és a kliens közötti kommunikáció során átvitt adatok mennyiségét jelenti egy másodpercre leosztva. Mivel minden kérés meghatározott hosszúságú, és a kérésekre adott válaszok hossza is állandó, ezért az adatátviteli sebességnek egyenes arányban kell állnia a sikeresen teljesített kérések számával a mérés során.

**Erőforrás igény** Az erőforrásigény a processzorhasználatból, és a felhasznált memória mennyiségéből áll össze. A mérés során a processzorhasználatnak követnie kell a feldolgozott kérések számát, hiszen az egyes kérések kiszolgálásához számítási kapacitás kell. A memóriaigény tekintetében nehéz megjósolni a mérés eredményét, mert nem ismerjük az alkalmazás szerverben alkalmazott memóriakezelésre vonatkozó stratégiákat. Például a belső gyorsítótárazás is nehézségeket okozhat.

## 3.4. A mérés menete

### 3.4.1. A RAP szerver oldalának mérése

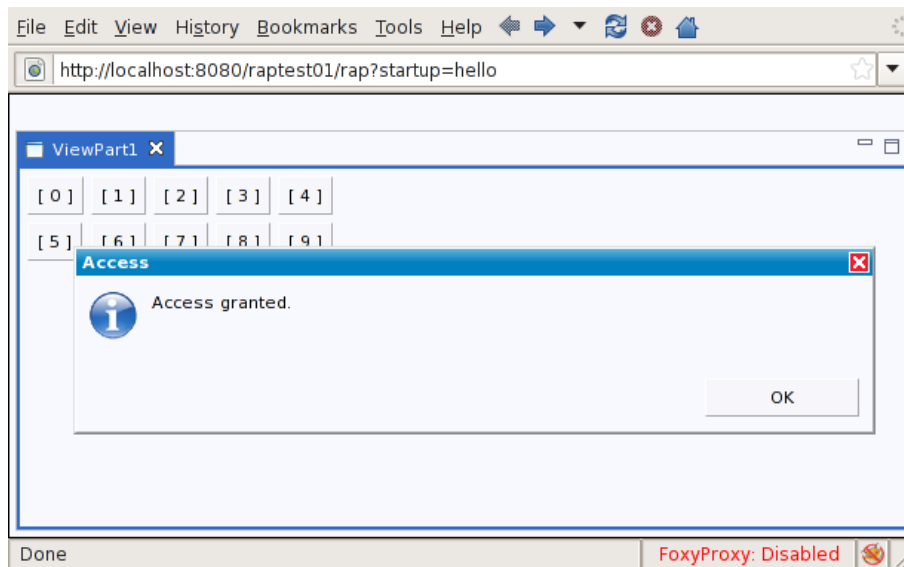
Maga a mérés a JMeter használatánál leírt módon zajlott. A méréshez egy általam készített RAP alkalmazást használtam, mint megfigyelt webes alkalmazást. Ahhoz, hogy az alkalmazás tesztelhető legyen, telepíteni kell azt az alkalmazás szerverbe. A szakirodalom ezt a lépést deploy néven illeti, és meg kell említeni, hogy RAP alkalmazásoknál ez nem triviális feladat.

Az alkalmazáson több méréssorozatot is végrehajtottam. Az első méréssorozatban változott a szerver által kapott összes kérés száma, a második sorozatban ez állandó volt. A méréssorozatok elemeit különböző számú felhasználóval végzett mérések jelentették.

A JMeter proxy szerverével végrehajtottam egy lépéssorozat rögzítését, majd visszajátszottam a forgalmat nagy mennyiségben. Eközben mértem a szerver terhelését, a kliens oldalon pedig maga a JMeter gyűjtötte a tárgyalt paramétereket, vagyis a késleltetést, a kérések feldolgozási sebességét, valamint a hálózati forgalmat.

### 3.4.2. A RAP kliens oldalának mérése

A kliens oldal mérése különböző böngészőkkel történt. Minden böngészőnél végrehajtottam ugyanazt a lépéssorozatot a kifejlesztett teszt-RAP alkalmazással. A mérés célja a jelenleg Li-



3.4. ábra. A tesztalkalmazás

nuxon elérhető böngészők összehasonlítása volt abból a szempontból, hogy melyik böngésző milyen erőforrás igényel hajtja végre az adott lépéssorozatot.

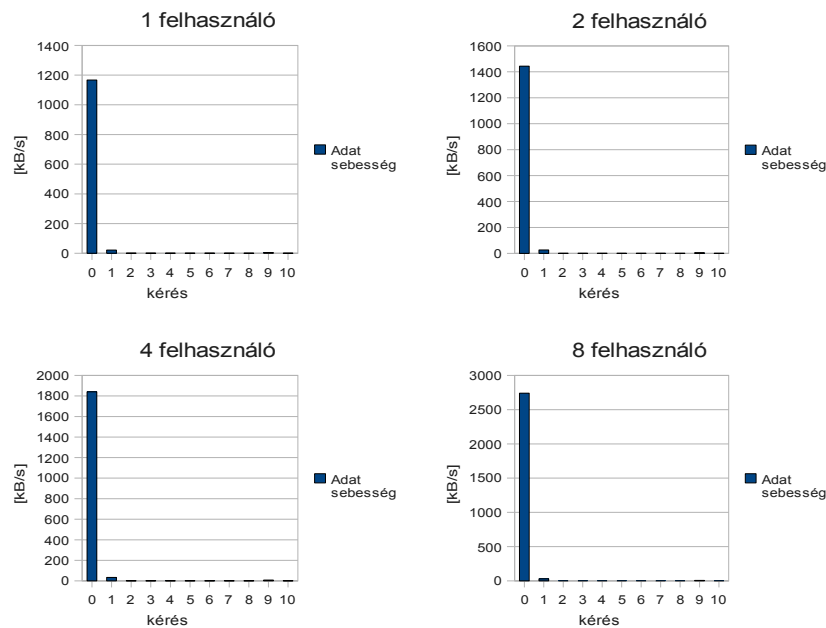
## 3.5. Mérési eredmények és azok analízise

### 3.5.1. A szerver oldal

A szerver oldal méréséhez tartozó Rich AJAX Platform alkalmazás néhány egyszerű felhasználói esemény beérkezésekor egy adatbázis művelettel reagál. Az alkalmazás egy gombokat tartalmazó nézetből áll, mely gombokat jó sorrendben megnyomva egy rekord kerül beszúrára az adatbázisban. Az alkalmazás felülete a 3.4 ábrán látható a helyes kombináció megadását követően.

A rögzített lépéssorozat visszajátszására a JMeter többféle konfigurációt biztosít. Beállítható az, hogy egy felhasználót reprezentáló szál hányszor futtassa le a lépéssorozatot, és hogy hány darab ilyen szál fusson párhuzamosan. Az első mérésben minden szál 20-szor ismételte meg a sorozatot, tehát a felhasználók számának növelésével a szerverre jutó összes kérések száma növekedett.

A második mérés esetében kíváncsi voltam a szerver terhelésének alakulására az eltérő felhasználószámoknál abban az esetben, ha a szerverre jutó összes kérések száma állandó a felhasználószám függvényében. A JMeter-t úgy állítottam be, hogy összesen mindig 400 kérést küldjön, azaz változtattam a felhasználónkénti kérések számát.



3.5. ábra. Adatátviteli sebesség alakulása az egyes kéréseknél

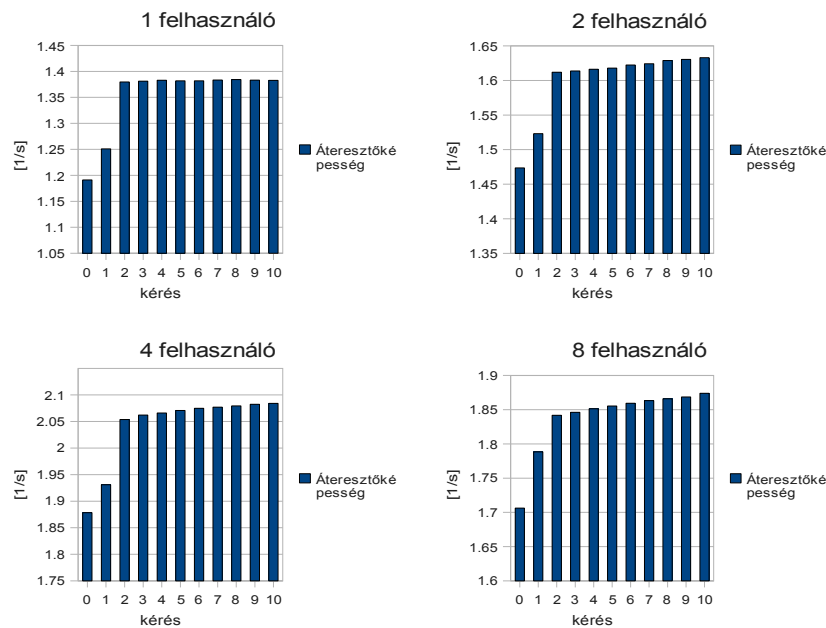
### Változó számú kérést tartalmazó mérésorozat

A mért adatátviteli sebességet a 3.5 ábra mutatja. Az ábrán négy grafikon látható, melyek az adatátviteli sebességeket ábrázolják a lépéssorozat egyes kéréseire.

Jól látható, hogy a forgalom nagy részéért a legelső kérés, illetve az arra érkező válasz felelős a felhasználók számától függetlenül. A Rich AJAX Platform működéséből adódóan a legelső kérés juttatja el a kliens oldali kódot a böngészőbe. Itt egy terjedelmes JavaScript kódról van szó, mely az egész qooxdoo könyvtárat tartalmazza. Ennek mérete egy megabyte környékén van a jelenlegi verziónál, de a kommunikációban a szerver tömörítést alkalmaz, ezzel a valóban átvitt adatmennyiség az első üzenetváltáskor megabyte helyett mindössze pár száz kilobyte terjedelmű, köszönhetően a jó tömörítési aránynak, amire a gzip szövegfájlok esetén képes.

A grafikonokról az is leolvasható, ahogyan az átviteli sebesség növekszik a felhasználók számának növelésének hatására. Annak, hogy az átviteli sebesség nem követi arányosan a felhasználói számot az az oka, hogy a nagyobb felhasználószám hatására a szerver lassabb működést produkál. Ha a szerver a kéréseket olyan ütemben tudná kielégíteni, ahogyan kisebb terhelésnél, úgy az adatátviteli sebesség hűbben tudná követni a felhasználószámot alakulását. Hasonló tendenciát mutat az áteresztő képesség menete a 3.6 ábrán. Az áteresztő képesség a nagyobb kéréseknél értelemszerűen kisebb, hiszen a terjedelmesebb adathalmazt több idő eljuttatni a klienshez. A második üzenetváltás hozza jelentősen kisebb az elsőnél, a lépéssorozat további tagjainak hossza pedig már csak pár száz byte nagyságrendjébe esik.

A késleltetésekre a 3.7 ábrán láthatóan szintén hatással van az üzenetváltások hossza. Meg-



3.6. ábra. Áteresztőképesség alakulása az egyes kéréseknél

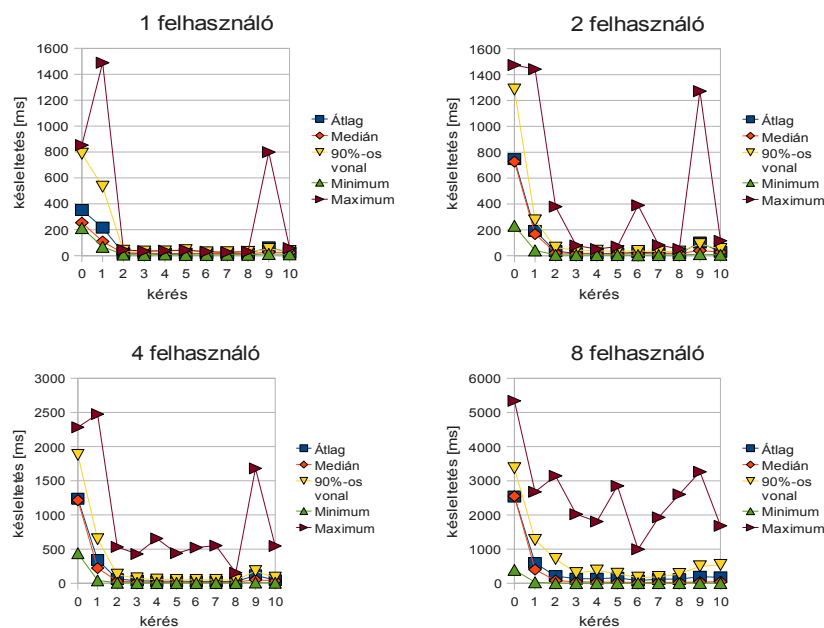
figyelhető, hogy az első két üzenetváltásnál mért késleltetésekhez képes a további üzenetek késleltetése sokkal kisebb.

Az előző grafikonokon láthattuk a mért paraméterek alakulását a kérések függvényében, a következőkben pedig a lépéssorozatokra összesített mérési eredmények lesznek ábrázolva a felhasználói szám függvényében. A vízszintes tengely logaritmikus beosztású. (3.8 ábra)

A 3.8(a) grafikonon ábrázolt adatátviteli sebesség függvényét az előzőekben tárgyaltak magyarázzák, vagyis annak okát, hogy az ábrázolt függvény miért marad el a lineáristól. Ha a szerver a megnövekedett felhasználói számmal lépést tudna tartani, akkor láthatnánk itt lineáris függvényt, vagyis a logaritmikus tengely beosztás esetén exponenciális képet.

A 3.8(b) grafikon az áteresztőképesség alakulását mutatja a felhasználói szám függvényében. A mért rendszer talán legfontosabb paramétere az áteresztőképesség, mert ez mutatja meg, hogy hány felhasználót tud az összeállított rendszer a legnagyobb sebességgel kiszolgálni. Ahhoz, hogy ezt megtudjuk, a görbe maximumát kell megkeresnünk. A jelen esetben alkalmazott mérési összeállítással ez valamivel 4 fölé esett. Hozzá kell tenni, hogy a szerveren egy végrehajtási szál futott, vagyis megállapítható, hogy egy szerveren futó kiszolgáló szál hozzávetőleg 4 klienst volt képes a legnagyobb áteresztőképességgel kiszolgálni.

Végül a 3.8(c) grafikonon a késleltetés alakulása figyelhető meg. Az ábra a vártnak megfelelő eredményt mutatja, tehát a késleltetés fokozatosan nő a felhasználók számával. Bár az átlagos késleltetés még 8 felhasználó esetén is egy másodperc alatti, tehát jónak mondható, ennek ellenére előfordulnak kiugrások, melyek miatt a maximális késleltetés már az 5 másodpercet



3.7. ábra. Késleltetés alakulása az egyes kéréseknél

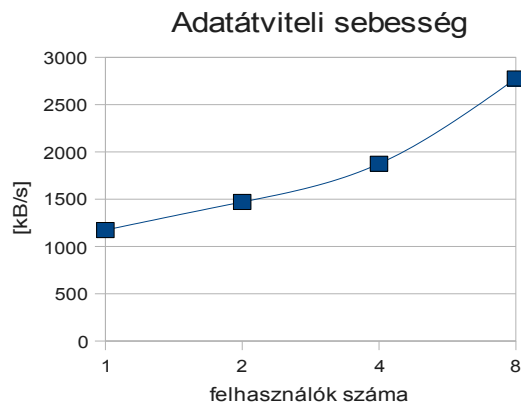
is eléri.

A szerver oldali teljesítménymérésének eredményét a 3.9 ábra mutatja. A processzorhasználatra vonatkozó mérések jól közelítették a várt eredményeket, viszont a memória használatról ugyanez nem mondható el. Főleg az alkalmazás szerver esetében tapasztaltam ellentmondásos értékeket, a memóriahasználat nem nőtt a felhasználók számával, sőt, több felhasználó esetén inkább kevesebb memóriát foglalt a szerver futás közben. A szerver memória foglalása 130-140 megabyte környékén mozgott, az adatbázis szerver beírta 33 megabyte-tal is.

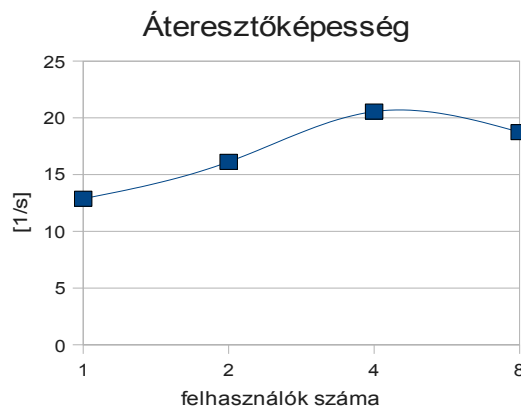
A processzorhasználat tekintetében már pozitívabb eredmények születtek. A kérések számának növekedésével a számítási igény nőtt, és ez meglátszik a 3.9(a) grafikonon. Az adatbázis szerver által produkált erőforrásigény elhanyagolhatóan bizonyult az alkalmazás szerveréhez képest, ami a mért alkalmazás alacsony adatbázis igényének tudható be.

### Állandó számú kérésből álló tesztsorozat

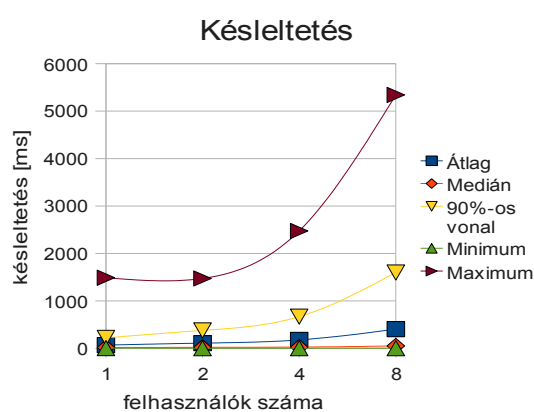
Ennél a mérésnél úgy került beállításra a terhelés, hogy minden felhasználószám esetén ugyanannyi számú kérés jusson a szerverre. A 3.10(a) grafikonon például az az eset látható, amikor összesen 400 lépéssorozat végrehajtási igény jut a szerverre, de a vízszintes tengelyen ezek más más felhasználói számnál jelentkeznek. Egy felhasználó esetén mind a 400 lépéssorozatot ugyanaz a felhasználó hajtja végre. Két felhasználónál mindkettő 200-200 lépéssorozat végrehajtását kezdeményezi egymással párhuzamosan, és így tovább. A grafikon tanúsága szerint



(a) Adatátviteli sebesség

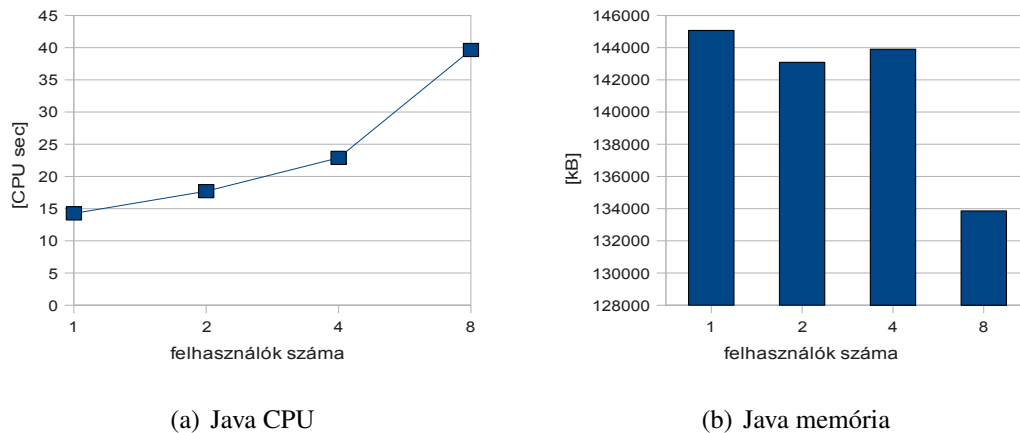


(b) Áteresztőképesség



(c) Késleltetés

3.8. ábra. Az első mérés eredményei



3.9. ábra. Az első mérés teljesítmény eredményei

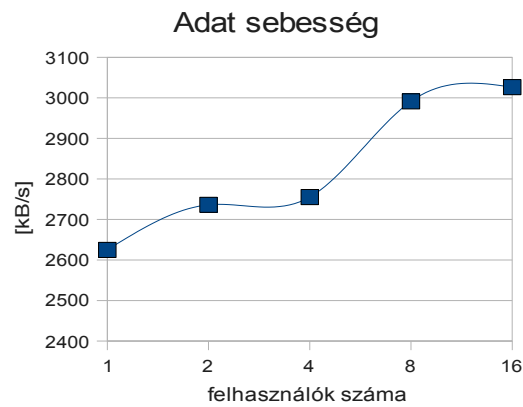
8 felhasználóig növekedés figyelhető meg, majd felette a növekedés jelentősen lelassul. Ennek okát az áteresztőképesség leromlásában kell keresni, ezt a 3.10(b) szemlélteti.

A grafikonról leolvasható, hogy áteresztőképességének a maximumát 8 felhasználó esetén éri el a rendszer. 8 felhasználó felett a kiszolgálás lelassul, és 16 felhasználót elérve visszaesik az egyedüli felhasználónál mért értékre.

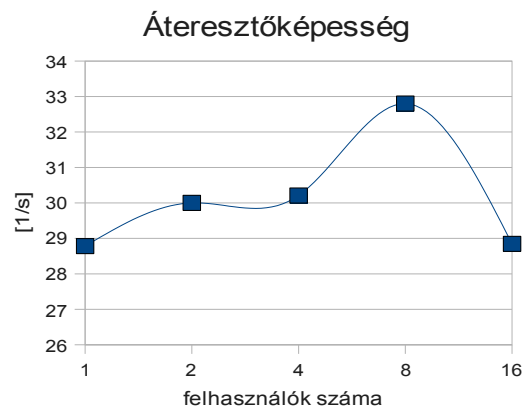
16 felhasználónál a késleltetés is kezd ingadozni, és átlagos értéke is eléri a fél másodpercet, a mért maximális késleltetés pedig 6 másodperc körüli.

A 3.11 ábrán látható a teljesítménymérés eredménye. Az alkalmazás szerver memóriaigénye 140-155 megabyte körül mozgott, de akár az előző mérésben, ennél sem követte a felhasználószám változását.

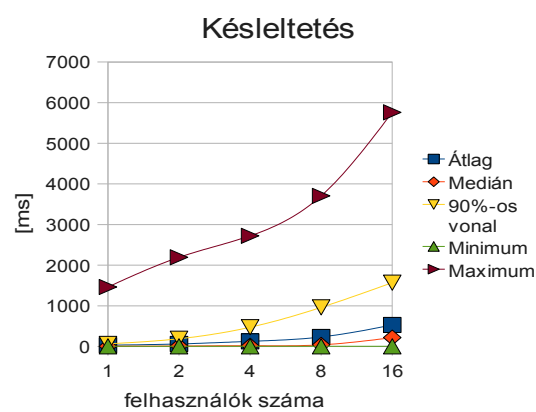
A processzorhasználat szempontjából az alkalmazás szerver az elvártaknak megfelelően teljesített. Mivel ebben a mérésben nem változott az összes beérkező kérés száma, várható volt, hogy a CPU használat sem fog számottevően változni. Mégis látható némi emelkedés a grafikonon, ami abból adódhat, hogy a párhuzamos kiszolgálás több kontextusváltást igényel a szerver alkalmazáson belül az egyes klienseket kiszolgáló programrészek között.



(a) Adatátviteli sebesség

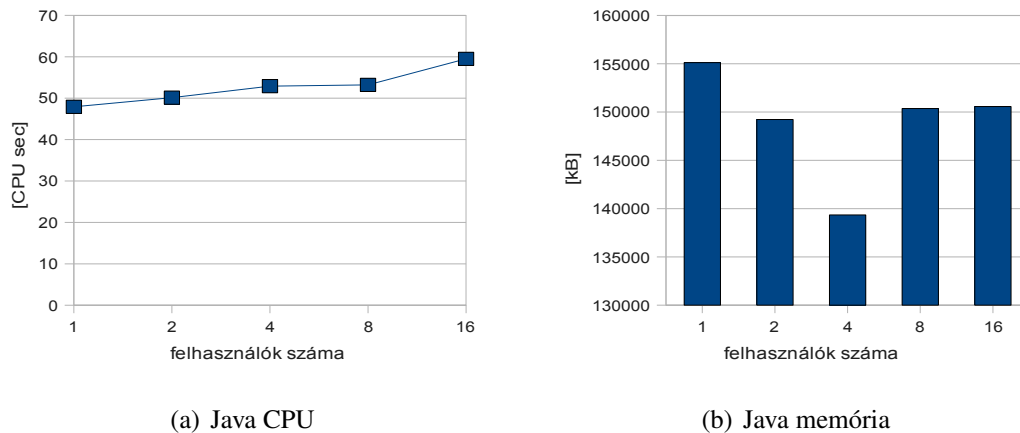


(b) Áteresztőképesség



(c) Késleltetés

3.10. ábra. A második mérés eredményei



3.11. ábra. A második mérés teljesítmény eredményei

### 3.5.2. A kliens oldal

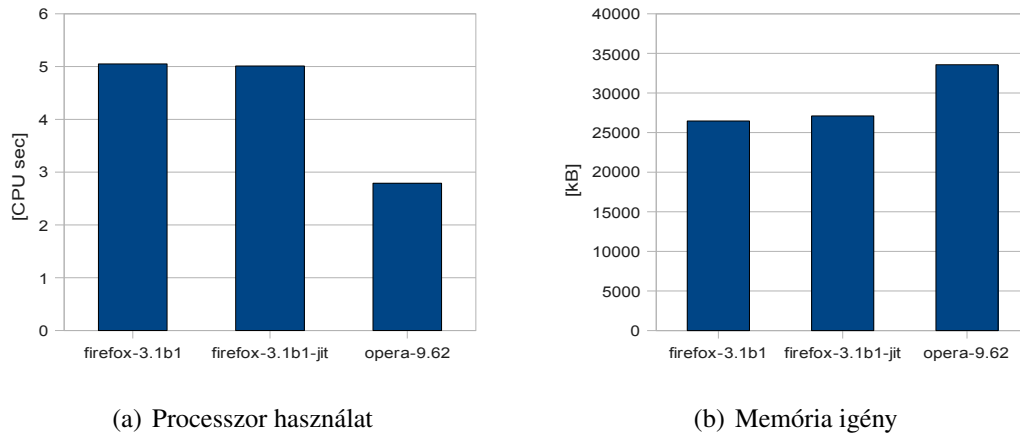
Érdekesképpen méréseket végeztem néhány Linuxon elérhető böngésző erőforrás-igény alakulását illetően Rich AJAX Platform alapú alkalmazások esetében. A 3.12 ábrán látható eredmények azt mutatják, hogy a vizsgált böngészők tekintetében nem a kliens oldal erőforrásigénye képezi a rendszer szűk keresztmetszetét, hiszen a szerver oldal által felhasznált processzor idő és memória is jelentősen nagyobb az itt tapasztaltaknál.

A mért böngészők közül az Opera igényelte a legkevesebb számítási erőforrást, nála azonban a Firefox memória használat szempontjából eredményesebb tudott lenni.

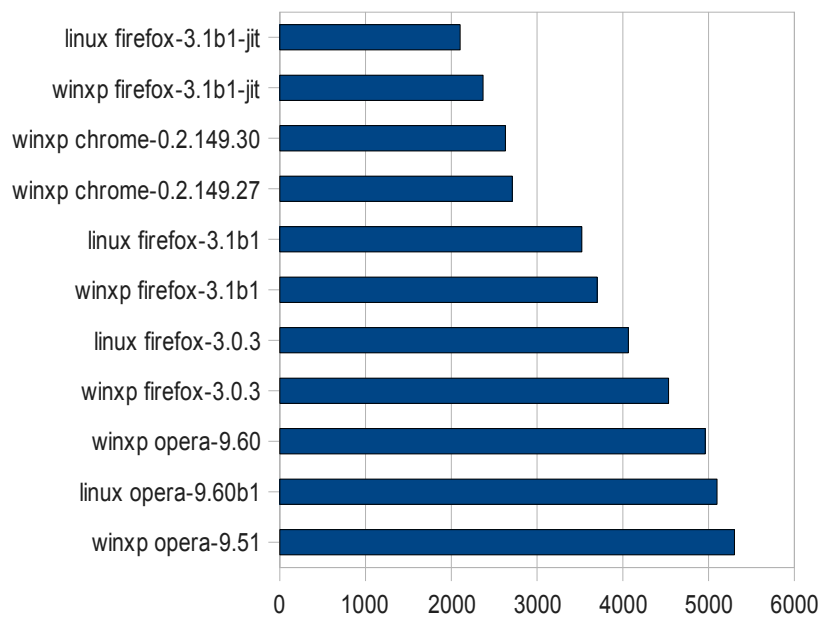
### SunSpider JavaScript benchmark

A népszerűbb böngészők közül néhányon lefuttattam a SunSpider nevű JavaScript benchmarkot, amely bizonyos algoritmusok JavaScript adaptációit futtatva méri azok végrehajtási idejét. A kapott eredményt a 3.13 ábra szemlélteti.

Megfigyelhető, hogy minden olyan böngésző, amely a JIT technológiát alkalmazza, megelőzi az összes olyan böngészőt, amely attól eltérő megoldással futtatja a JavaScript kódokat. Bár mint az előző, RAP-es kliens oldali mérésnél az látható volt, jelenleg nem a kliens oldali végrehajtás a Rich AJAX Platform szűk keresztmetszete, a JIT lehetővé teszi a kliens oldalon komplexebb algoritmusok futtatását.



3.12. ábra. Rich AJAX Platform - kliens oldali teljesítmény



3.13. ábra. A SunSpider JavaScript benchmark eredménye

## 4. fejezet

# Összefoglalás

A fejezet összefoglalja a dolgozatban leírt eredményeket és az elvégzett munkát, majd kitér a dolgozathoz kiegészítő elemekre is.

### 4.1. Elért eredmények

A célt a félév során az volt, hogy egy Rich AJAX Platform alapú webalkalmazások tesztelésére alkalmas környezetet alakítsak ki, és ebben teljesítmény orientált méréseket hajtsak végre. Az elérhető szoftverek összehasonlítása után egy Apache JMeter alapú tesztkörnyezetet állítottam össze, melynek minden komponense szabadon elérhető, nyílt forrású szoftverekből áll. A tesztkörnyezet segítségével mérhetőek a RAP alapú webalkalmazások legfontosabb teljesítmény paraméterei: a késleltetés, az áteresztőképesség, az adatátviteli sebesség, és a szerver oldali erőforrásigény. A környezet teljesítménymérés mellett alkalmas a szerver oldali alkalmazás korrekt működésének ellenőrzésére is.

### 4.2. Javítási, bővítési lehetőségek

- Az alkalmazások skálázódásának vizsgálata a felhasználói felület összetettségének függvényében.
- Kliens oldalon intenzív RAP alkalmazás fejlesztése a böngészők összehasonlítására.
- Komplex adatkötés hatásának vizsgálata az erőforrásigény szempontjából.
- Alternatív tesztelő program használata a kliens oldalon a JMeter helyett. A Grinder hatékony script felületet biztosít tesztek vezérlésére, ezzel komolyabb teszt esetek lennének kifejleszthetők.
- Selenium alkalmazása a böngészők erőforrásigényének vizsgálatára.

# Irodalomjegyzék

- [1] {<http://en.wikipedia.org/wiki/ApacheBench>}.
- [2] {<http://www.eclipse.org/>}.
- [3] Nick Edgar. *Eclipse Rich Client Applications*. 2004. {[http://www.eclipsecon.org/2004/EclipseCon\\_2004\\_TechnicalTrackPresentations/11\\_Edgar.pdf](http://www.eclipsecon.org/2004/EclipseCon_2004_TechnicalTrackPresentations/11_Edgar.pdf)}.
- [4] {<http://grinder.sourceforge.net/index.pdf>}.
- [5] {<http://en.wikipedia.org/wiki/IMacros>}.
- [6] **Just-in-time compilation**. {[http://en.wikipedia.org/wiki/Just-in-time\\_compilation](http://en.wikipedia.org/wiki/Just-in-time_compilation)}.
- [7] {<http://jakarta.apache.org/jmeter/>}.
- [8] {<http://wiki.eclipse.org/RAP/LoadTesting>}.
- [9] John Moore. *What is Ajax?* 2008. {<http://www.riaspot.com/articles/entry/What-is-Ajax->}.
- [10] {<http://www.osgi.org/About/Technology?section=2>}.
- [11] *Hypertext Transfer Protocol*, 1999. {<http://www.ietf.org/rfc/rfc2616.txt>}.
- [12] {<http://seleniumhq.org/>}.
- [13] Ryan Slobojan. *Eclipse Ganymede: An in-depth look at RAP (Rich Ajax Platform)*. 2008. {<http://www.infoq.com/news/2008/06/eclipse-ganymede-rap>}.
- [14] SunSpider JavaScript Benchmark. {<http://www2.webkit.org/perf/sunspider-0.9/sunspider.html>}.
- [15] Albert M.C. Tam. *Enabling Process Accounting on Linux HOWTO*, 2001. {<http://tldp.org/HOWTO/Process-Accounting/>}.

[16] {[http://en.wikipedia.org/wiki/Web\\_2.0](http://en.wikipedia.org/wiki/Web_2.0)}.

[17] David Mosberger és Tai Jin. **httperf - A Tool for Measuring Web Server Performance**, 1998. {<http://www.hpl.hp.com/research/linux/httperf/wisp98/httperf.pdf>}.