Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

TERMÉK ALAPÚ BEVÉTELMEGOSZTÁSOS ELSZÁMOLÓRENDSZER FEJLESZTÉSE

Készítette: Markert László András
Telefon: +36203429667
E-mail: laci@markert.hu

Konzulens: Szőllősi Loránd, TMIT

2007.

I. Kivonat

Napjainkban a tartalomkezelő rendszerek (Content Management System, CMS) reneszánszukat élik. Sokan azt hiszik, hogy amikor CMS-ekről beszélünk, akkor az internetes webportálok adatbázishátteréről van szó. Valójában azonban ez csak egy megjelenése a CMS-ek rendkívül szerteágazó és bonyolult változatainak. Az összes digitális tartalom elhelyezhető (és éppen ezért el is helyezik) CMS-ekben, amelyek aztán ezen tartalmak struktúrált, jogosultságrendszert is támogató tárolását és módosítását teszik lehetővé.

Amióta kereskedelem létezik, mindig problémát jelentett a pontos elszámolás. Nincs ez másképp a digitális tartalmak kereskedelmében sem. A termékek sajátosságából, illetve a többszereplős piaci környezetből adódóan alakult ki a digitális világban is a bevételmegosztásos elszámolás. Azonban az adatforgalom soha nem látott méretű növekedése olyan alapvető problémákat hozott a felszínre, ami miatt teljesen új technológiájú elszámolórendszerek fejlesztésére van szükség.

Egy új, puha valós idejű elszámolórendszer tervezésénél figyelembe kell venni a korábbi rendszereknél felmerült problémákat, és lehetőség szerint az összeset el is kell kerülni. Fontos, hogy a rendszer önálló, problémamentes működése nem garantálja a végső sikert: fel kell készülni a tökéletes együttműködésre a különböző tartalomkezelő rendszerekkel. A minden részletre kiterjedő tervezést követheti a rendszer tényleges megvalósítása.

Az új elszámolórendszer „piacra dobását” meg kell hogy előzze a teljeskörű tesztelés. Szükséges a tervezés és a megvalósítás során elkövetett hibák elkülönítése, kijavítása, valamint a más rendszerekkel történő összehasonlítás.

A tesztelés befejeztével meg lehet vizsgálni, hogy a létrehozott rendszer milyen további funkciókkal bővíthető, mi az, amit még javítani lehet rajta. Egy informatikai rendszer esetében ez kiemelt fontossággal bír, hiszen a digitális világ fejlődése egyre gyorsabb és gyorsabb.

II. Abstract

Content Management Systems (CMS) are reviving nowadays. Many people consider CMS as equivalent with database background of webportals. But as a matter of fact this is only one of the many elaborate variations of CMS applications. All the digital contents can be and are even enclosed in CMS, that allows one to store and modify these contents in a well arranged and permission supporting way.

Exact settling of accounts is a problem since trading has started. The same holds true of trading in digital contents. In the digital world the speciality of the products and the market have created the settling of accounts based on revenue division. However the huge growth of data traffic has brought key problems to surface. This is the reason why account systems based on a brand new technology have to be developed.

When planning a new soft real-time account system problems that came up at former systems need to be considered and avoided if possible. It is important, that the perfect and independent running of the system does not indicate the final success, because it also has to collaborate with various Content Manager Systems. The detailed planning can be followed by the implementation of the system.

Before being placed on the market the new account system needs to be entirely tested. Failures of planning and implementation need to be separated and corrected. The new account system needs to be compared with other systems.

After testing can be checked whether the system can be improved and completed with further functions. This is of crucial importance in case of a software system, since the digital improvement is going on and on.

III. Tartalomjegyzék

I. Kivonat I
II. Abstract II
III. Tartalomjegyzék III
IV. Ábrák jegyzéke V
V. Táblázatok jegyzéke VI
1. Bevezetés 1
2. Tartalomkezelő rendszerek megismerése 4
2.1. Tartalomkezelő rendszerek és az Internet 4
2.2. Tartalomkezelő rendszerek a mobil világban 8
3. Bevételmegosztásos rendszerek problémái 12
3.1. Bevételmegosztásos rendszerek 12
3.2. Állomány alapú elszámolórendszer 14
3.2.1. Tesztadatok előállítása 15
3.2.2. Az állomány alapú tesztrendszer felépítése 15
3.2.3. Az állomány alapú rendszer tesztjének eredménye 16
3.3. Adatbázis alapú elszámolórendszer 18
3.3.1. Az adatbázis alapú tesztrendszer felépítése 19
3.3.2. Az adatbázis alapú rendszer tesztjének eredménye 20
3.4. Az állomány és az adatbázis alapú rendszerek összehasonlítása 23
4. Puha valós idejű elszámolórendszer tervezése 26
4.1. Elvárások 26
4.2. A rendszer követelményei 27
4.2.1. Hardverkövetelmények 28
4.2.2. Szoftverkövetelmények 28
4.3. A rendszer funkciói 28
4.3.1. Tranzakciók fogadása 29
4.3.2. Betöltés 29
4.3.3. Kimentés 30
4.3.4. Elszámolás 31
4.4. A rendszer működése 31
4.4.1. Alapvető függvények 33
5. Bevételmegosztásos elszámolórendszer fejlesztése 35

5.1. Fejlesztői környezet bemutatása ... 35
5.1.1. Hardverkörnyezet ... 35
5.1.2. Szoftverkörnyezet ... 36
5.2. Belső függvények ... 37
5.2.1. Tömb- és faátalakító függvények ... 38
5.2.2. Tranzakciókezelő függvények ... 39
5.2.3. Ki- és bemeneti adatokkal kapcsolatos függvények ... 41
5.2.4. Rendszerbeállítási függvény ... 43
5.2.5. Elszámolással kapcsolatos függvények ... 44
5.2.6. Megjelenítéssel kapcsolatos függvények ... 47
5.2.7. Egyéb függvények ... 48
5.3. Összefüggések a rendszer függvényei között ... 49
5.4. Tranzakciókezelő modul ... 50
5.5. Elszámoló modul ... 52
5.6. Biztonsági mentés, havi mentés, betöltés ... 55
5.7. Külső kapcsolódási pontok ... 56
5.7.1. Tranzakciók átadása az elszámolórendszernek ... 57
5.7.2. Tranzakciók generálása ... 59
5.8. Biztonsági kérdések ... 60
5.8.1. Állományolvasás ... 61
5.8.2. Kommunikáció ... 62
5.9. Hibakezelés ... 63
6. Tesztelés ... 65
6.1. Összehasonlítás a tesztrendszerekkel ... 66
6.2. Összehasonlítás egy működő rendszerrel ... 68
7. Továbbfejlesztési lehetőségek ... 71
7.1. Kompromisszumok a fejlesztés során ... 71
7.2. Továbbfejlesztési lehetőségek ... 73
8. Összegzés ... 76
VI. Függelék ... VII
VII. Irodalomjegyzék ... IX
VIII. Rövidítések ... XI
IX. Köszönetnyilvánítás ... XIII

IV. Ábrák jegyzéke

2.1. ábra: Az ARPANET kiépítettsége 1977. július 15-én [10] 5
2.2. ábra: Internet-felhasználók száma kontinensek szerint [14] 6
2.3. ábra: Internetes hosztok számának alakulása 1994 és 2007 között [11] 6
2.4. ábra: Mobiltelefon-felhasználók megoszlása régiónként, GSM/UMTS elterjedtsége [23] 9
2.5. ábra Mobiltelefon-kapcsolatok számának alakulása az elmúlt öt évben [8] . 10
3.1. ábra: Az első teszt eredményei (állomány alapú tesztrendszer) 16
3.2. ábra: A második teszt eredményei (állomány alapú tesztrendszer) 17
3.3. ábra: A harmadik teszt eredményei (állomány alapú tesztrendszer) 18
3.4. ábra: Az első teszt eredményei (adatbázis alapú tesztrendszer) 20
3.5. ábra: A második teszt eredményei (adatbázis alapú tesztrendszer) 21
3.6. ábra: A harmadik teszt eredményei (adatbázis alapú tesztrendszer) 22
3.7. ábra: Tíz véletlenszerűen kiválasztott tartalom letöltési számának meghatározása.. 23
3.8. ábra: Tíz új tranzakció elkönyvelése.. 24
4.1. ábra: Az elszámolórendszer működése 33
5.1. ábra: Az aggregator() függvény működése 46
5.2. ábra: A rendszer függvényeinek kapcsolata.. 49
5.3. ábra: A buffer() függvény működése 59
5.4. ábra: A konfigurációs állomány felépítése 61
6.1. ábra: Az első teszt eredményei (memória alapú elszámolórendszer) 66
6.2. ábra: A második teszt eredményei (memória alapú elszámolórendszer) 67
6.3. ábra: A harmadik teszt eredményei (memória alapú elszámolórendszer) 67

V. Táblázatok jegyzéke

2.1. táblázat: Az ARPANET történetének mérföldkövei 5
3.1. táblázat: A bevételmegosztásos piaci példa szereplői 13
5.1. táblázat: A tesztrendszer hardveregységei 35
5.2. táblázat: A tesztrendszer szoftveregységei 36
5.3. táblázat: Az egyes lekérdezések és az aggregator() függvény hozzájuk tartozó válasza 45
5.4. táblázat: A CDR felépítése 57
6.1. táblázat: Az egyes teszteredmények összehasonlítása 68

1. Bevezetés

Az Internet és a mobilkommunikáció elterjedésével a tartalomkezelő rendszerek minden korábbinál nagyobb számban terjedtek el a világban. A jelenleg létező weboldalak nagyobbik része, illetve a mobiltelefonos tartalmakat nyújtó szolgáltatások szinte kivétel nélkül mind-mind valamilyen tartalomkezelő rendszert alkalmaznak a felhasználók kiszolgálására.

A tartalomkezelő rendszerek elterjedésével új távlatok nyíltak meg az elszámolórendszerek előtt is. Gyakorlatilag minden termék alapú tartalomkezelő rendszer esetén szükség lehet egy elszámolórendszerre, ami a vásárlásokat (tranzakciókat) elkönyveli, és a pénzügyi időszak (általában egy hónap) végeztével elszámolást készít a kereskedés résztvevői számára.

Az elszámolást nem csak a tartalmak mennyisége és a forgalom nagysága nehezítheti, hanem az elszámolásban érdekelt szereplők száma is. A mobiltelefonos tartalmak piaca például tipikusan egy olyan sokszereplős ágazat, ahol egyetlen tartalom bevétele akár hat vagy még annál is több szereplő között oszlik meg. Ilyen környezetben fokozott jelentőséggel bírnak az elszámolórendszerek.

Az elszámolórendszerekben jelenleg alkalmazott megoldások általában egy állományt (állomány alapú elszámolórendszerek) vagy egy adatbázist (adatbázis alapú elszámolórendszerek) helyeznek a rendszer középpontjába, a tranzakciók fogadását és elkönyvelését, illetve az elszámolás készítését e köré szervezik. Ez a megközelítés sokszor hibás, mivel nem alkalmazkodik az adatstruktúra a tartalmak speciális felépítéséhez (a hasonló tartalmak egymástól „fizikailag” távol helyezkednek el), így mind az elszámolás, mind pedig a tranzakciókezelés nehézkes.

Szükség van az elszámolórendszerek felépítésének és kifejlesztésének teljes újragondolására, és egy újfajta rendszer létrehozására, ami segítséget jelent a

fenti probléma megoldásában. A probléma elsősorban abban csúcsosodik ki, hogy a nem megfelelően megtervezett elszámolórendszer futási ideje hatalmasra növekedhet.

A feladatom az, hogy ezt az új, puha valós idejű, termék alapú, bevételmegosztást is támogató elszámolórendszert megtervezzem és kifejlesszem, figyelembe véve a jelenleg létező rendszerek problémáit és a piaci igényeket. Célom, hogy a tervezés állomásait, a fejlesztés lépéseit, illetve a már kész rendszer tesztelését és továbbfejlesztési lehetőségeit részletesen bemutassam.

A 2. fejezetben megismerkedünk a tartalomkezelő rendszerek általános fogalmával. Röviden bemutatom a tartalomkezelő rendszerek nélkülözhetetlenségéhez vezető utat, illetve a jelenleg létező és gyakran alkalmazott megoldásokat.

A 3. fejezetben ismertetem a bevételmegosztás fogalmát és az ezzel kapcsolatos tudnivalókat. Néhány teszten és példán keresztül megismerkedünk továbbá a bevételmegosztásos rendszerek problémáival egy állomány alapú és egy adatbázis alapú tesztrendszeren keresztül.

A 4. fejezetben előtérbe kerül az új bevételmegosztásos rendszer, azon belül is a tervezés lépéseit fogom részletesen bemutatni. Egyúttal általános célkitűzéseket és konkrét elvárásokat fogok megfogalmazni, amiket az új rendszernek teljesítenie kell majd.

Az 5. fejezetben az elszámolórendszer fejlesztését fogom részletesen ismertetni. Az egyes megvalósított modulok és funkciók részletezése mellett szó lesz a különböző mérnöki döntésekről is, amik a fejlesztés során születnek. Külön hangsúlyt fektetek a rendszerszintű működés bemutatására.

A 6. fejezetben a megvalósított rendszer teljeskörű tesztelését fogom elvégezni. Ekkor kiderül, hogy a célkitűzések közül melyeket sikerült megvalósítani, és mennyire sikerült megfelelni az előzetes elvárásoknak. Összehasonlítási alapként a tesztrendszereket és egy már használatban lévő elszámolórendszert fogok használni.

A 7. fejezetben ismertetem, hogy a rendszert miként és hogyan lehet további funkciókkal bővíteni, kiegészíteni. A továbbfejlesztési lehetőségek között mind egyszerűbb, mind bonyolultabb megoldásokat is felsorolok. A fejlesztés során meghozott kompromisszumokat is bemutatom.

Végül a 8. fejezetben összegzem a tapasztalataimat, amelyeket a rendszer tervezése, fejlesztése és tesztelése során szereztem.

2. Tartalomkezelő rendszerek megismerése

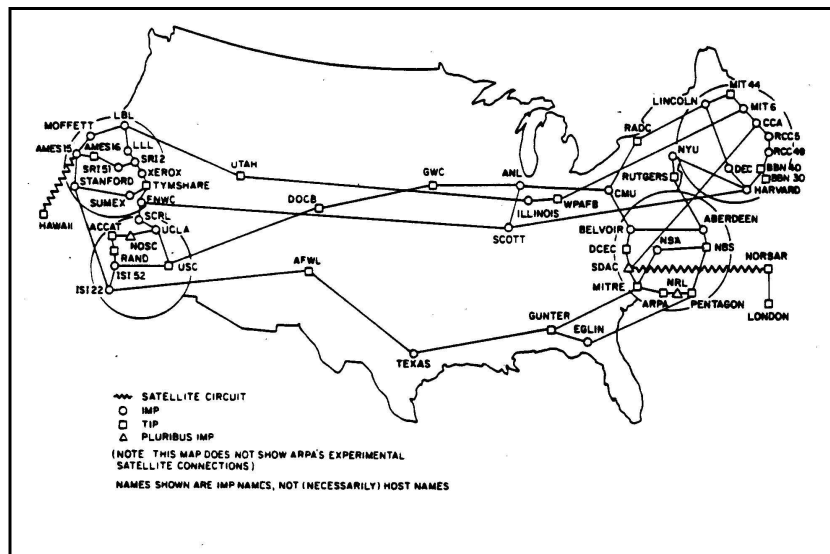
A tartalomkezelő rendszer (Content Management System, CMS) olyan szoftverrendszer, amely nem strukturált információk elkészítését, kezelését és tárolását segíti. A CMS gondoskodik a tartalmak struktúrált megjelenítéséről, statisztikák készítéséről, valamint nagyfokú támogatást nyújt kiegészítő funkciók integrálásához. [22]

A tartalomkezelő rendszerek kialakulása szoros összefüggésben van az Internet elterjedésével. A rengeteg, hatalmas méretű és bonyolultságú weblap megjelenése szükségessé tette, hogy olyan szoftverrendszereket fejlesszenek ki, amik segítenek az információk teljeskörű kezelésében. A következőkben röviden bemutatom, hogy hogyan alakult ki az a környezet, ami nélkülözhetetlenné tette a tartalomkezelő rendszereket.

2.1. Tartalomkezelő rendszerek és az Internet

Mint sok minden más, az Internet is egy katonai fejlesztésnek köszönheti létrejöttét. A Hidegháború idején a Szovjetunió előnybe került az Amerikai Egyesült Államokkal szemben az Űrversenyben (1957, Szputnyik-1). Emiatt Dwight D. Eisenhower, az USA akkori elnöke elrendelte a DARPA létrehozását, ami a különböző kutatások finanszírozását és koordinálását látta el. Felmerült az igény, hogy az ország különböző részein található kutatóközpontok és egyetemek között legyen egy elosztott, csomagkapcsolásos hálózat, ami egy esetleges atomtámadás során sem omlana össze elosztott jellegéből adódóan. Így született meg 1969-ben az ARPANET, amit nyugodtan nevezhetünk a mai Internet ősének.

Az ARPANET fejlődése töretlenül folytatódott a '70-es években is. A kezdeti négy ARPANET-csomópontot (SRI, UCLA, UCSB, UTAH) gyorsan követte sok kutatóközpont és egyetem az USA egész területén, majd később Európában is. A hálózatot a katonai kutatásokhoz kapcsolódó feladatokon kívül már a korai években is használták állományok cseréjére, elektronikus levelezésre és távoli bejelentkezésre, illetve a csomagkapcsolt adattovábbítás kutatására. Az ARPANET kiépítettségének 1977-es állapotát a 2.1. ábra szemlélteti. [2], [20], [21]



2.1. ábra: Az ARPANET kiépítettsége 1977. július 15-én [10]

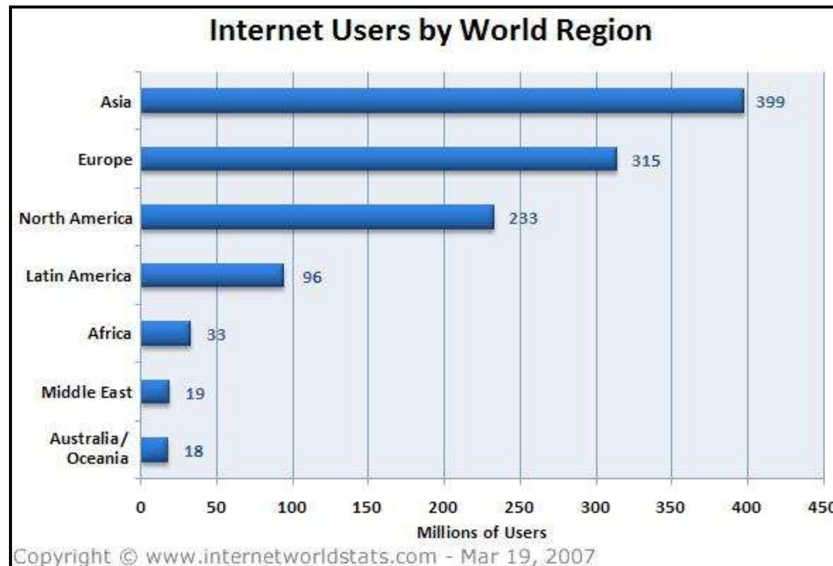
Az ARPANET történetének legfontosabb mérföldköveit tartalmazza a 2.1. táblázat.

Évszám	Esemény
1969	Az ARPANET megszületése
1974	Az Internet kifejezés első felbukkanása
1983	A MILNET kiválik az ARPANET-ből
1985-86	Az ARPANET és az NSFNet összekapcsolása
1989	Az ARPANET megszűnése

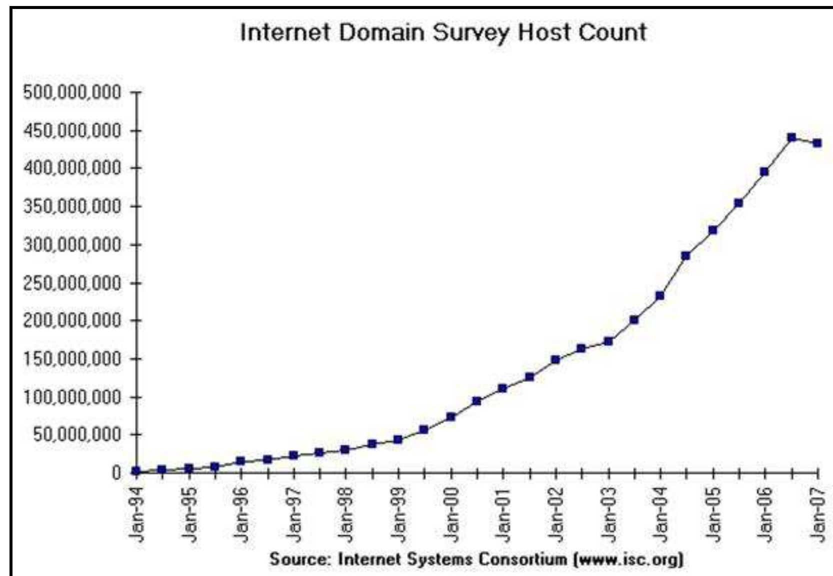
2.1. táblázat: Az ARPANET történetének mérföldkövei

Az ARPANET-et és az Internetet egyaránt jellemzi az, hogy időről-időre olyan technológiák bukkannak fel, amik a rajtuk folyó forgalmat hatalmas mértékben megnövelik. Az ilyen technológia az ún. killer application. Az első ilyen alkalmazás az e-mail volt (1972), ezt követte a web (1991), majd a p2p (1999).

Az ilyen megoldások minden esetben segítenek abban, hogy az Internet növekedése újabb lendületet vegyen. [5]



2.2. ábra: Internet-felhasználók száma kontinensek szerint [14]



2.3. ábra: Internetes hosztok számának alakulása 1994 és 2007 között [11]

Az Internet főleg a 2000-es évektől kezdődően hatalmas fejlődésen ment keresztül. Az Internet-felhasználók száma (2.2. ábra), a weboldalak mennyisége (a hosztok számát mutatja a 2.3. ábra) és a forgalom nagysága napról-napra nő, újabb és újabb rekordokat dönt meg. Azonban a hatalmas méretű információhalmaz, amit a Web számunkra jelent, mit sem ér anélkül, hogy azt

rendszerezetten, könnyen megtalálhatóan elérnénk. Azok a technológiák, amelyek néhány éve még jól működtek, és megfelelő szolgáltatást nyújtottak a felhasználóknak, jelenleg már idejétmúlttá váltak, elavultak. A statikus weblapok majdnem egy évtizeden keresztül teljes mértékben kielégítették a piaci igényeket, az elmúlt néhány évben viszont olyan szinten elterjedtté vált az Internet, hogy egy magára valamit is adó cég vagy szervezet nem elégedhet meg egy ritkán frissülő weblappal. A tartalomkezelő rendszerek erre a problémára is megoldást nyújtanak.

Az évek során rengeteg különféle webes CMS született. Sokféle megközelítéssel, technológiával és minőséggel találkozhatunk az ilyen rendszereket vizsgálva, azonban mindegyik megvalósítás közös jellemzője az alapvető CMS-jelleg. Ezen CMS-eket tehát egyaránt jellemzi a struktúrálatlan információk teljeskörű kezelésének támogatása, illetve a struktúrált megjelenítés minél szélesebbkörű biztosítása.

A különféle webes CMS-ek hatalmas száma azzal jár, hogy sokszor nehéz az igényekhez legjobban igazodó rendszert kiválasztani. Lehetőség van ingyenes vagy fizetős rendszereket választani, technológiák (ASP, PERL, JAVA, PHP – ez utóbbi a legelterjedtebb) alapján is meghozható a döntés, illetve sok ingyenes rendszer ki is próbálható [15]. Természetesen egy vállalati tartalomkezelő rendszer bevezetésénél a fenti szempontok egyáltalán nem biztosítják a sikert. Egy komplex, intranetes megoldásokat, fokozott biztonságot és a hatalmas terhelés elbírását garantáló rendszer kiválasztása és a vállalat igényeihez való igazítása hosszadalmas és bonyolult folyamat. Komoly tanulmányok foglalkoznak a piacon levő CMS-ek összehasonlításával, elsősorban nagyvállalatok szempontjai és igényei alapján. [3]

Nehéz becslést adni arra, hogy napjainkban hány webes CMS található az Interneten. A különböző, szabadon felhasználható vagy fizetős CMS-ekből jelenleg egyes vélemények szerint több ezer megoldás létezik, azaz ennyi a száma azoknak a rendszereknek, amelyekből – elterjedtségüktől függően – néhány tucattól akár több millióig terjedő számú oldal található [6]. Ezekhez kell még hozzávenni a részben vagy teljesen saját fejlesztésű rendszereket. Ha a CMS-ek palettáját a

nagyvállalati intranetes megoldásoktól egészen a legegyszerűbb blogig vizsgáljuk, akkor nagyságrendileg százmillió rendszerről beszélhetünk. Ez elsőre túlzásnak tűnhet, de ha belegondolunk, hogy az Interneten barangolva szinte minden oldal használ valamilyen CMS-jellegű szolgáltatást, akkor e becslés reális számként értékelhető. Az internetes hosztok számával (2.3. ábra) összevetve látható, hogy a CMS-ek vagy a CMS-jellegű megoldásokat használó oldalak szinte az egész Internetre kiterjednek. [18]

A fenti becslés természetesen pontatlan, illetve minden tartalomkezelést használó oldalra vonatkozik. Ennél kevesebb a ténylegesen tartalomkezelő rendszert alkalmazó, valamilyen termék kereskedelmével foglalkozó portál, azaz opcionálisan elszámolást igénybevevő weboldal. Azonban az internetes hosztok, illetve a felhasználók számából könnyen lehet arra következtetni, hogy a tartalomkezelő rendszerek, illetve a hozzájuk kapcsolódó szolgáltatások jelentős üzleti értékkel bírnak. Itt is nehéz számszerűsíteni azt, hogy a különböző weboldalakon keresztül éves szinten mekkora forgalmat bonyolítanak (azaz mekkora piacot jelent az e-kereskedelem), azonban néhány felmérés alapján ez az összeg akár több száz milliárd dollárra is rúghat [16]. Ekkora piacon igenis szükség van elszámolórendszerekre.

2.2. Tartalomkezelő rendszerek a mobil világban

A mobiltelefonos tartalmakkal kapcsolatos CMS-ek bemutatásához röviden összefoglalom, hogy milyen út vezetett a mobilkommunikáció, és ezáltal a mobiltelefonos tartalmak elterjedéséhez.

Az első generációs (1G) mobiltechnológia az 1980-as évek elején jelent meg. A rendszer analóg volt, és országonként különböző megvalósítások születtek. A szabványosítást 1982-ben a CEPT-GSM kezdte meg, majd az ETSI folytatta. Az 1G-s rendszerek kizárólag beszédátvitelre voltak alkalmasak.

A második generációs (2G, GSM) mobiltechnológia 1990-ben jelent meg, és napjainkban is ez a legelterjedtebb mobiltelefonos rendszer. Nagy előnye az 1G-

hez képest, hogy ez már az egész világon szabványosított, digitális rendszer volt, amely nagyobb adatátviteli sebességével, illetve egy felhasználó számára több csatorna kiosztásával sokkal jobb szolgáltatás nyújtására volt alkalmas.

Az ún. 2,5G további előrelépést jelentett a 2G után: a GSM vonalkapcsolt működésével szemben nagy előny volt a csomagkapcsolt megvalósítás. A sebesség GPRS-technológiánál négyszerese, EDGE-technológiánál majdnem húszszorosa a 2G-s rendszerek sebességének.

A harmadik generációs (3G, UMTS) mobiltechnológia az utóbbi években kezd nagymértékben elterjedni. Előnye, hogy ez egy ténylegesen világméretű rendszer, így a készülékek mindenhol működőképesek (a korábbi rendszereknél – főleg az analóg 1G esetén – ez korántsem volt így), sebessége pedig akár több mint százszorosa is lehet a 2G-s technológia sebességének. [19]

Jelenleg a 2G és a 3G uralják a mobilkommunikációs világot. Egyes felmérések alapján jelenleg a mobiltelefon-felhasználók 84%-a ezeket a technológiákat használja (2.4. ábra).

2.4. ábra: Mobiltelefon-felhasználók megoszlása régiónként, GSM/UMTS elterjedtsége [23]

Már a Kivonatban is említettem, közkeletű félreértés az, hogy a CMS-eket a webportálokkal azonosítják. Való igaz, hogy a webportálok a CMS-ek

legelterjedtebb előfordulásai, ám emellett számtalan más felhasználási területen is alkalmazzák. Egyik ilyen a mobiltelefonos tartalmak (napjainkban: háttérképek, animációk, MP3 csengőhangok, játékok, néhány évvel ezelőtt: operátorlogók, mono- és polifónikus csengőhangok) kezelése.

Ahogy a webes CMS-ek kialakulásánál és elterjedésénél is nagy szerepet játszott a felhasználók számának gyors növekedése, nincs ez másképp a mobiltelefonos rendszereknél sem. Jelenleg nagyjából 1,1 milliárd felhasználója van az Internetnek. A mobiltelefon-kapcsolatok számából (2.5. ábra) következtetve megállapíthatjuk, hogy majdnem kétszer annyi mobiltelefon-felhasználó van, mint Internet-felhasználó.

2.5. ábra Mobiltelefon-kapcsolatok számának alakulása az elmúlt öt évben [8]

A mobiltelefonos tartalmakat szolgáltató tartalomkezelő rendszereket vizsgálva megállapíthatjuk, hogy e kevésbé szem előtt lévő alkalmazásokban legalább olyan bonyolult és kifinomult megoldásokat alkalmaznak, mint a legösszetettebb webes CMS-eknél. A következőkben az ilyen tartalmak üzleti jelentőségét mutatom be.

A mobiltelefonos tartalmakból származó bevételek 2006-ban nagyjából 19 milliárd dollárt tettek ki világszerte, és az előrejelzések szerint a piac öt év alatt megduplázódik. A legnépszerűbb tartalmak a különböző csengések, zenék és képek, amelyek az egész forgalom majdnem kétharmadát teszik ki. A mobiltelefonos tartalmak szinte kizárólag CMS-eken keresztül kerülnek értékesítésre, így a fenti összeg felfogható a mobiltelefonos CMS-ek éves forgalmaként is. [4]

A mobiltelefonos tartalmak értékesítése napjainkban egyre inkább az Interneten, azon belül a WAP segítségével történik. A mobiltelefon-felhasználó telefonja segítségével különböző WAP-oldalakat keres fel (leggyakrabban saját mobilszolgáltatójának oldalát), ahol a mobiltelefonos tartalmak között böngészhet, illetve akár azonnal meg is vásárolhatja azokat. A fizetés számlás előfizetés esetén az egyenlegének terhére, más esetben (pl. feltöltőkártyás előfizetés) valamilyen más keret segítségével történik. A választott tartalom sokféleképpen kerülhet a felhasználó telefonjára, az esetek legnagyobb részében SMS vagy MMS technológián keresztül megkapja, vagy letöltheti az adott terméket.

A CMS-ek az imént bemutatott szolgáltatásnál nélkülözhetetlenek. A különböző WAP-oldalakat CMS-ben tárolják. A tartalmak leírását, árát, a támogatott készülékek listáját és magát a tartalmat is CMS-ben helyezik el. A hozzáférések szabályrendszerét szintén a CMS segítségével állítják be. A vásárlásokat pedig a CMS fogadja, és ezután a szolgáltatás letöltését is a CMS teszi lehetővé. Ezt a folyamatot nem befolyásolja az, hogy a mobiltelefonos szolgáltató, vagy egy attól független cég szolgáltatását nézzük. Megállapíthatjuk, hogy a CMS-ek a mobil világban hatalmas szerepet játszanak, és így a különböző elszámolórendszerek is kiemelt fontossággal bírnak.

3. Bevételmegosztásos rendszerek problémái

Az előző fejezetben megismerkedtünk a napjainkban rendkívül elterjedt tartalomkezelő rendszerekkel. Azonban a tartalomkezelés önmagában nem elég: szükség van a tartalmak eladásával összefüggő feladatok ellátására is. Erre nyújtanak megoldást az elszámolórendszerek.

Elszámolórendszerből – ahogy tartalomkezelő rendszerből is – rendkívül sokféle megoldás létezik. A legáltalánosabb elszámolórendszerek, amelyek különleges funkciókkal vagy szolgáltatásokkal nem rendelkeznek, esetünkben nem relevánsak. A következőkben a bevételmegosztásos elszámolórendszerekkel fogunk megismerkedni, majd pedig az elszámolórendszerek jelenleg elterjedt technológiáit ismertetem.

3.1. Bevételmegosztásos rendszerek

A digitális tartalmak kereskedelménél – a piac és a termékek sajátosságai miatt – egyre elterjedtebb a bevételmegosztásos elszámolás alkalmazása. Egy ilyen terméknél a bevételeket a hagyományos termelő-kereskedő vagy termelő-nagykereskedő-kiskereskedő viszonylathoz képest esetenként jóval több szereplő között is el kell osztani. A piacon nem ritka az akár hat vagy hét szereplő között megoszló bevétel sem. Ilyenkor a hagyományos kereskedelemben működő elszámolások csak nagyon nehezen vagy sehogy sem működnek.

Erre a problémára nyújt megoldást tehát a bevételmegosztás (revenue sharing, RS). A felhasználó vásárlásakor befolyó összeget szerződések és jogszabályok alapján osztják szét a piaci résztvevők. A legegyszerűbb a rendszer működését egy példán keresztül bemutatni.

Adott egy digitális termék. Ezt a terméket valaki előállította (egy konkrét mobiltelefonos példánál: a háttérképet megrajzolta), ő a tulajdonos. Ha ez a termék nem egy önmagában álló entitás, hanem például egy adott tulajdonos hasonló témájú termékei közé illeszkedik, akkor tartozik a termékhez egy márkanév, amit az angol nyelvből vett brand szóval jelölünk. Egy adott országban egy adott brandet általában egyetlen cég értékesíthet, amit a brand tulajdonosával kötött megállapodás garantál számára. Előfordulhat, hogy egy-egy brandet vagy termékcsoportot viszonteladók is értékesíthetnek. Természetes szereplője a piacnak az a cég is, amelyik a termék kereskedelmét tulajdonképpen elvégzi (mobiltelefonos példánál: mobilszolgáltató, másnéven operátor). Olyan tartalomnál, amelyet valamilyen jogszabály vagy egyéb megállapodás alapján szerzői jog véd, további szereplőként feltűnhet egy jogvédő iroda is. Passzív résztvevőként pedig minden esetben részesül a bevételből az állam, valamilyen forgalmi adó formájában. A példa szereplőit foglalja össze a 3.1. táblázat.

Piaci szereplő	Rövid leírás
Tulajdonos	A tartalom előállítója, tulajdonosa
Jogtulajdonos	Az adott termék (vagy brand) eladási jogát birtokolja
Viszonteladó	Továbbértékesítési joga van
Operátor	A kereskedelem keretfeltételeit biztosítja
Jogvédő iroda	Jogvédett termék esetében szerepel
Állam	Forgalmi adó formájában részesül a bevételből

3.1. táblázat: A bevételmegosztásos piaci példa szereplői

A fenti piacon a szereplők között bonyolult szerződéses megállapodások és jogszabályi viszonyok vannak. E keretekben meghatározható, hogy az adott termék eladásából származó bruttó bevétel a szereplők között milyen formában oszlik el.

A bevételmegosztásos piacot további speciális feltételek árnyalhatják. Egyik ezek közül az ún. sávos bevételmegosztási modell, aminek lényege az, hogy az előbbi példában említett bevételmegosztás nem független a forgalom nagyságától (vagy más paraméterektől), hanem annak megfelelően változik.

A következőkben konkrét elszámolórendszerekkel ismerkedünk meg. Mindegyik bemutatott rendszer működhet bevételmegosztásos alapon, így elsősorban a technológiai alapokat mutatom be.

3.2. Állomány alapú elszámolórendszer

A legegyszerűbb, legáltalánosabb rendszer az állomány alapú elszámolórendszer. A különböző jogszabályi és szerződési kötelezettségek miatt a tranzakciók egyetlen szöveges állományban kerülnek eltárolásra, amit ún. chargelognak nevezünk. A chargelog soronként értelmezhető, minden egyes sor egy-egy tranzakciónak felel meg. Tartalmazza – többek között – a tranzakció időbélyegét, a vásárló azonosítóját és egyéb attribútumait, illetve a vásárolt tartalom számtalan tulajdonságát. Egy eladott tartalom egyértelműen hozzárendelhető a bevételmegosztás egy-egy résztvevőjéhez, kezdve az operátornál, egészen a jogtulajdonosig. A chargelog ilyen felépítése lehetővé teszi tehát, hogy a szöveges állomány tartalmát végigolvasva, illetve néhány egyéb adat segítségével az elszámolás elvégezhető legyen.

Az állomány alapú elszámolásnak egyetlen alapvető hibája van: nem erre való. A chargelog elsősorban adminisztratív okok miatt készül, az ez alapján folyó elszámolás egyáltalán nem optimális és meglehetősen lassú. Maga a rendszer egy állományra épül, az elszámolás készítése során azonban nem csak az állományt használja, hanem különböző egyéb adatokat is. Általános megoldás az, hogy a chargelog mellett egy adatbázisban tárolják ezeket az adatokat, a chargelog feldolgozása során pedig az éppen szükséges információkat az adatbázisból kérdezik le (például az adott tartalom árát, ami a chargelogban nem szerepel). Ez a megoldás további hátulütőket is hordoz magában. A lekérdezések során akár több táblára is szükség lehet egyidejűleg a megfelelő adatok előállításához. Ilyenkor pedig – főleg nagymennyiségű adat esetén, ami esetünkben teljes mértékben fennáll – gyakran bekövetkezik az, hogy a sok lekérdezés és táblakapcsolás (ún. join) miatt a rendszer sebessége drasztikusan visszaesik (erről részletesen a 3.4. részben lesz szó).

Az állomány alapú elszámolórendszer alapvető hibáinak bizonyítására elkészítettem egy tesztrendszert, hogy az elszámoláshoz szükséges alapvető funkciókat megvalósítva megmérhessem az azokhoz szükséges időket.

3.2.1. Tesztadatok előállítása

A rendszer megvalósítása előtt a tesztekhez szükséges adatokat állítottam elő. A tesztadatok generálásánál figyelni kellett arra, hogy az adatok a valóságnak nagyjából megfelelőek legyenek, mind felépítés szempontjából, mind pedig számosságukat tekintve. A tesztadatok generálásához egy egyszerű függvényt írtam, amely a PHP beépített mt_rand() függvényét használta a véletlen értékek előállítására.

Egy generált tranzakció egy időbélyeget, egy független és négy összefüggő attribútumot tartalmazott. A tranzakciókat egy egyedi azonosító jelölte. Ilyen tranzakcióból ötszázat generált a függvény, mindegyikhez tartozott egy szám is, amely minimum 10, maximum 100 volt. Egy ilyen tranzakció – szám páros tehát tekinthető egy adott mobiltelefonos tartalomnak, és egy adott hónapban az ezen tartalom letöltései számának.

Mivel az adatokat különböző tesztrendszerekbe kellett betölteni, elvégeztem rajtuk néhány apróbb módosítást. Az állomány alapú tesztrendszer lényegében egy chargelog, azaz a tranzakciók egyesével, soronként kerülnek rögzítésre egy állományban. Így a generált tranzakció – szám párosokat úgy kellett átalakítani, hogy a számnak megfelelő darabszámú tartalom legyen. Azonban az életszerűség és a pontosabb teszteredmények érdekében ehhez a burst-jelleget is ki kellett küszöbölni, azt a látszatot keltve, hogy a tranzakciók (ahogy az a valóságban is tapasztalható) megfelelően elkeveredve kerüljenek a chargelogba. Az adatbázis alapú tesztrendszernél ilyen átalakításra nem volt szükség.

3.2.2. Az állomány alapú tesztrendszer felépítése

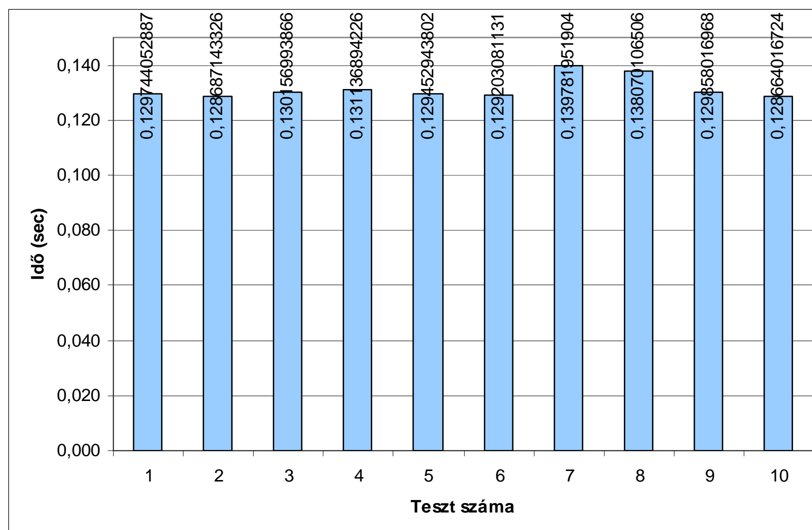
Az állomány alapú elszámolórendszer középpontja maga a chargelog. Ebből kifolyólag minden ezzel kapcsolatos funkció tulajdonképpen az állomány írását és olvasását, illetve a beolvasott adatokon végzett műveleteket jelenti.

Az állományhoz való hozzáféréseket a PHP beépített állománykezelő függvényei (fopen(), feof(), fgets(), fputs(), fclose() stb.) segítségével végeztem.

A chargelog olvasásra és hozzáfűzésre került megnyitásra, írásra nem. A teszt előtt az előző részben bemutatott tesztadatokat töltöttem be az állományba.

3.2.3. Az állomány alapú rendszer tesztjének eredménye

Az első tesztnél a feladat tíz véletlenszerűen kiválasztott tartalom letöltési számának meghatározása volt. Az egyes tartalmakat egyesével, egymás után összegezte a rendszer, és az egyes összegzésekhez szükséges időket mérve lehetett a rendszer gyorsaságát vizsgálni. A teszt időeredményeit a 3.1. ábra tartalmazza.

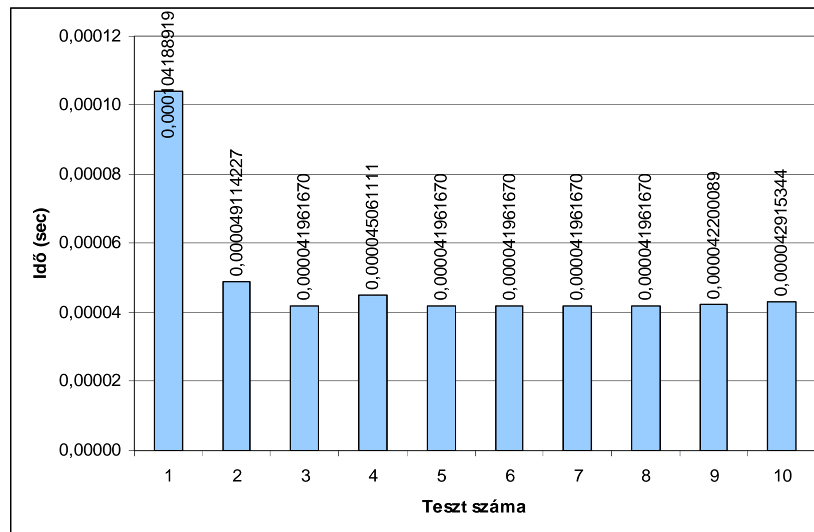


3.1. ábra: Az első teszt eredményei (állomány alapú tesztrendszer)

Látható, hogy az egyes tartalmak összegzése tizedmásodperces nagyságrendű. A legkisebb futási időt (0,129 sec) a második esetnél, a legnagyobbat (0,140 sec) a hetedik esetnél mértem, az átlagos idő 0,131 sec, 0,004 sec szórással. A hetedik és nyolcadik eset kiugró időeredményének oka valószínűleg valamilyen egyéb folyamat. Ha azokat nem vesszük figyelembe, az átlag számottevően akkor sem módosul.

A második teszt során az újonnan beérkező tranzakciók elkönyvelését teszteltem. A chargelog felépítéséből adódóan (minden tranzakció külön sor) ezen tesztnél nem kellett külön vizsgálni az olyan tartalmakat, amik egyszer már

szerepelnek az állományban azoktól, amelyek még nem. A mérés tulajdonképpen egy állománykezelésből és -írásból állt. Az ezekhez szükséges időket tartalmazza a 3.2. ábra.

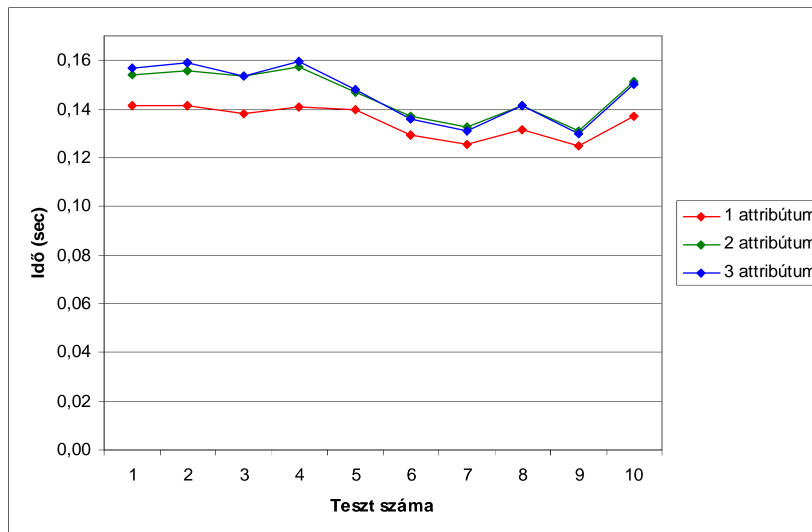


3.2. ábra: A második teszt eredményei (állomány alapú tesztrendszer)

Az új tranzakciók elkönyvelése – egy esettől eltekintve – tíz mikroszekundumos nagyságrendű idő alatt megtörtént. A legkisebb futási időt (42 μsec) a tesztesetek felénél, a legnagyobbat (104 μsec) az első esetnél mértem, az átlagos idő 49 μsec, 19 μsec szórással. Az első eset kiugróan magas időeredményt produkált, valószínűleg utána az állományhoz való hozzáférés cache-eléssel történt, ez okozta a nagymértékű javulást.

A harmadik teszt során azt vizsgáltam, hogy egy rendkívül egyszerű elszámolás-jellegű feladat lefutása mennyi időt vesz igénybe. Ezen tesztelszámolás során az egyes mobiltartalmak nem összes attribútumuk alapján kerültek szummázásra, hanem csak egy, kettő vagy három előre lerögzített attribútum szerint. Ez a teszt nem túl pontosan, de jellegében az igazi elszámoláshoz hasonlóan működött, hiszen az elszámolás során is bizonyos attribútumokat figyelembe veszünk, másokat pedig nem. (Egyetlen fontos dolog maradt ki az elszámolásból: a különféle egyéb adatok begyűjtése más adatbázisból vagy állományból. Ez egy jelentéktelen problémának tűnik, de a későbbiekben bemutatom, hogy miért ez az egyik legnagyobb hibaforrás az elszámolások

készítése során.) Mind a három alteszt eseteinek időeredményeit tartalmazza – és ennek segítségével összehasonlíthatóak a próbaelszámolásokhoz szükséges időeredmények a rögzített attribútumok számának függvényében – a 3.3. ábra.



3.3. ábra: A harmadik teszt eredményei (állomány alapú tesztrendszer)

A tesztesetek futási ideje tizedmásodperces nagyságrendű. A legkisebb futási időt (0,125 sec) az egy attribútumos kilencedik esetnél, a legnagyobbat (0,160 sec) a három attribútumos negyedik esetnél mértem, az átlagos idő 0,143 sec, 0,011 sec szórással. A tesztek azért mozogtak az ábrán látható módon együtt, mert az attribútumok növelését az előző teszt – az egy attribútummal kevesebbet használó eset – felhasználásával végeztem. (Ez azonban nem befolyásolta a futási idők nagyságrendjét.)

3.3. Adatbázis alapú elszámolórendszer

Egy adatbázis alapú elszámolórendszernél a tranzakciók és az azokhoz tartozó tartalomletöltések száma egy adatbázisban kerülnek eltárolásra. A korábban már említett chargelogra ennél az esetnél is szükség lehet, azonban itt ezen állomány az elszámolás készítésekor semmilyen funkciót sem tölt be, csak

különböző szabályzati okokból van rá szükség. A chargelog az esetleg előforduló rendszerhibák vagy reklamációk esetén nyújthat segítséget.

Ennél a rendszernél az adatbázis felépítése magában hordozza a tranzakciók strukturális tulajdonságait, míg a tranzakciós adatokat az adatbázis egy-egy sorában tárolhatjuk. Ez a megoldás nagyobb gyorsaságot és hatékonyságot ígér az állomány alapú rendszerhez képest, mivel egy tartalomhoz minden esetben hozzátartozik az aktuális letöltésszám, így nem kell külön erre is összegezni a sorokat.

3.3.1. Az adatbázis alapú tesztrendszer felépítése

A teszthez létrehoztam egy MySQL-adatbázist, és abban elkészítettem egy, a tesztadatok befogadására alkalmas táblát. A táblában nem tároljuk el a tranzakció időbélyegét, viszont attribútumain kívül még egy oszlopot kell felvennünk: a letöltések számát. A tranzakció beérkezésekor a tartalmat egyértelműen meghatározó tulajdonságok alapján megvizsgáljuk, hogy a táblában szerepel-e már ilyen tartalmat kezelő sor. Amennyiben igen, úgy az adott sor letöltési számát tartalmazó mezőt inkrementáljuk. Ha még nem érkezett eddig ilyen tartalom, akkor egy új sort hozunk létre, amiben a most érkezett tranzakció tulajdonságait, illetve a letöltéseinek számát tároljuk (ami ekkor értelemszerűen 1 lesz – feltételezve, hogy egy tranzakció egy letöltésnek felel meg). A fenti megoldás létrehozásához tehát először egy lekérdezésre van szükség, amikor megvizsgáljuk, hogy szerepel-e már a beérkezett tranzakcióhoz tartozó tartalom az adatbázisban.

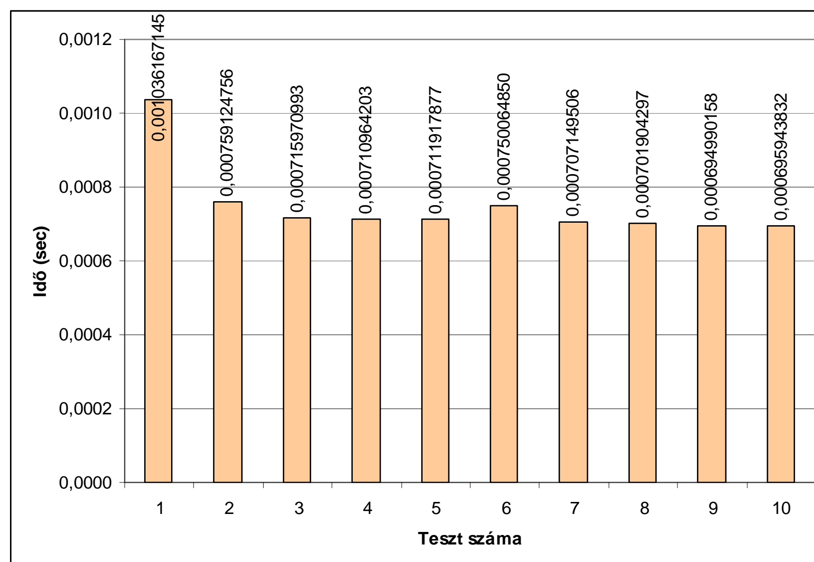
Az elszámolási időszakban egy ilyen tábla minden pillanatban az addig megvásárolt tartalmak számát teljesen egyértelműen megadja. Akár végzetes hiba esetén is lehetőség van a rendszer újraindítására, ekkor a tranzakciókat a chargelogból lehet kinyerni, azonban ez egy rendkívül időigényes feladat lehet, hiszen a chargelogot soronként kell újrafeldolgozni. Amíg hibamentes futás esetén ez egy hónapban nagyjából egyenletesen elosztott adatbázis-műveleteket jelent, addig a chargelogból átvett tranzakciók nem igényelnek túl sok időt. Ha

viszont valamilyen hiba folytán a hónap teljes korábbi forgalmát kell újrafelépíteni az adatbázisban, az sokkal időigényesebb feladat.

A tesztrendszer működéséhez elsősorban a PHP beépített MySQL-függvényeit használtam. Egy adott teszt többnyire egy lekérdezésből, majd az adatbázisból érkező adatok kezeléséből, illetve az azokon elvégzett műveletekből állt. Az egyes tesztek időigényét vizsgáltam, és ezek alapján értékeltem az adatbázis alapú tesztrendszert.

3.3.2. Az adatbázis alapú rendszer tesztjének eredménye

Az első tesztnél a feladat tíz véletlenszerűen kiválasztott tartalom letöltési számának meghatározása volt. A feladat pusztán a meghatározott paramétereket kielégítő sor letöltési számot tartalmazó oszlopának visszaadása volt. A teszt időeredményeit a 3.4. ábra tartalmazza.

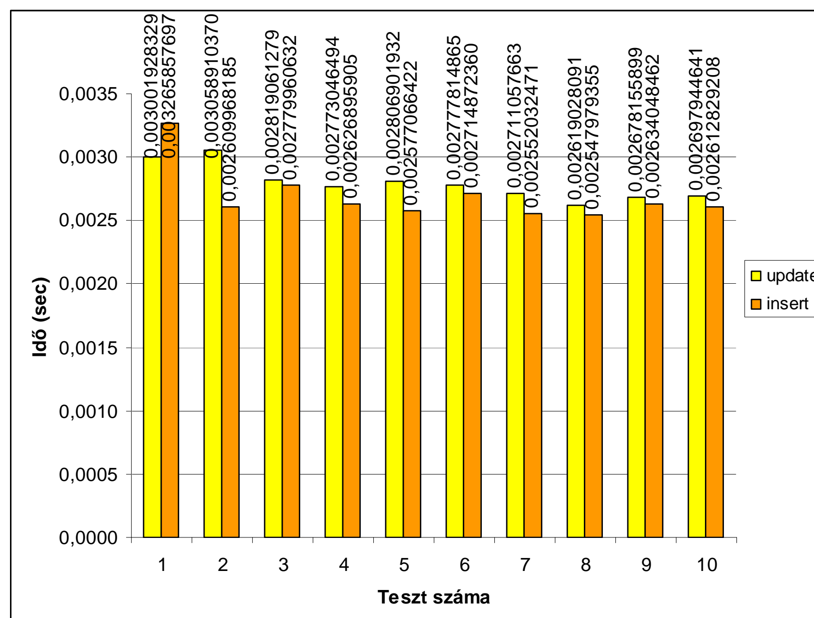


3.4. ábra: Az első teszt eredményei (adatbázis alapú tesztrendszer)

Az egyes tartalmak letöltési számának lekérdezése – egy esettől eltekintve – száz mikroszekundumos nagyságrendű. A legkisebb futási időt (694 μsec) a kilencedik esetnél, a legnagyobbat (1036 μsec) az első esetnél mértem, az átlagos idő 748 μsec, 103 μsec szórással. Az első eset kiugró időeredményét követő

gyorsulás a MySQL működéséből adódik, valószínűleg a memóriába kerülő adatbázis-tartalom csökkentette a mért időket.

A második teszt során az újonnan beérkező tranzakciók elkönyvelését teszteltem. Mivel külön esetnek számít az, amikor már szerepelt az adatbázisban az éppen beérkező tartalom attól, mint amikor még nem, ezért mind a két lehetőségnek megfelelő teszteseteket használtam. Ez utóbbi probléma általában az adatbázis-kezelő rendszerekre érvényes: ha egy sor egy celláját szeretnénk megváltoztatni, akkor először ellenőriznünk kell, hogy az adott sor létezik-e. Amennyiben nem létezik, fel kell vennünk az adott sort, mert az adatbázis-kezelő nem képes a módosításra, ha nincsen az adott keresési feltételre illeszkedő sor (módosításnál először egy keresés fut le, és annak eredményeire alkalmazza a módosítási műveleteket a rendszer). A 3.5. ábra mind az adatbázisban már létező tartalom letöltési számának inkrementálását (update), mind az új sor beszúrását (insert) mutatja.

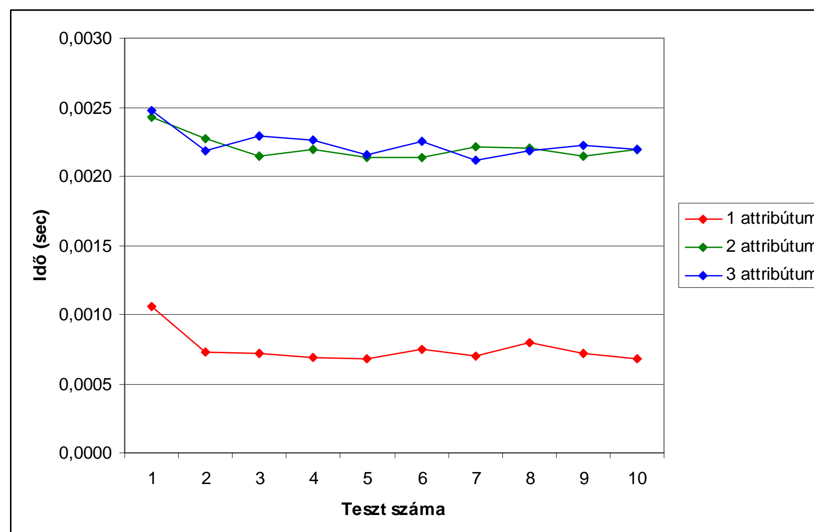


3.5. ábra: A második teszt eredményei (adatbázis alapú tesztrendszer)

Az új tranzakciók elkönyvelése ezredmásodperces nagyságrendű idő alatt megtörtént. A két aleset (update, insert) között számottevő időkülönbség nem volt mérhető, egy esettől eltekintve azonban mindig az update műveletnek volt nagyobb az időigénye. A legkisebb futási időt (0,00254 sec) a nyolcadik insertnél,

a legnagyobbat (0,00327 sec) az első insertnél mértem, az átlagos idő 0,00274 sec, 0,00018 sec szórással. A kiugró első esetek utáni gyorsulás valószínűleg a MySQL korábban említetett cache-eléséből adódnak.

A harmadik teszt során azt vizsgáltam, hogy egy rendkívül egyszerű elszámolás-jellegű feladat lefutása mennyi időt vesz igénybe. Ezen tesztelszámolás során az egyes mobiltartalmak nem összes attribútumuk alapján kerültek szummázásra, hanem csak egy, kettő vagy három előre lerögzített attribútum szerint. Ez a teszt nem teljesen pontosan, de jellegében az igazi elszámoláshoz hasonlóan működött, hiszen az elszámolás során is bizonyos attribútumokat figyelembe veszünk, másokat pedig nem, és így számoljuk ki az egyes tartalomcsoportok számát. (Ennél az elszámolásnál is figyelmen kívül hagytam az egyéb adatok elérését, ami komoly hatással van a tényleges elszámolások időeredményeire, ám jelen esetben ezzel nem foglalkoztam, a későbbiekben még lesz róla szó.) A 3.6. ábra mind a három alteszt eseteinek időeredményeit tartalmazza, így összehasonlíthatóak a különböző számú paraméterek szerinti elszámolások időigényei.



3.6. ábra: A harmadik teszt eredményei (adatbázis alapú tesztrendszer)

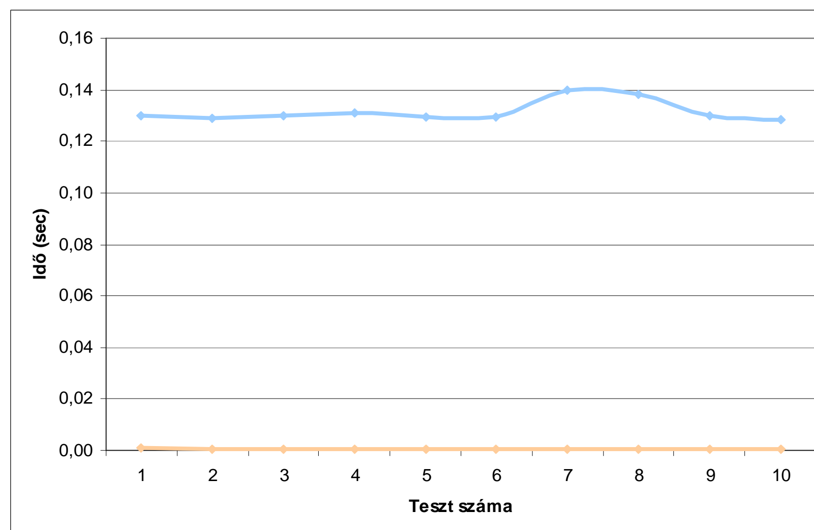
A tesztesetek futási ideje legfeljebb ezredmásodperces nagyságrendű volt. A legkisebb futási időt (0,00068 sec) az egy attribútumos ötödik esetnél, a

legnagyobbat (0,00248 sec) a három attribútumos első esetnél mértem, az átlagos idő 0,00173 sec, 0,00071 sec szórással.

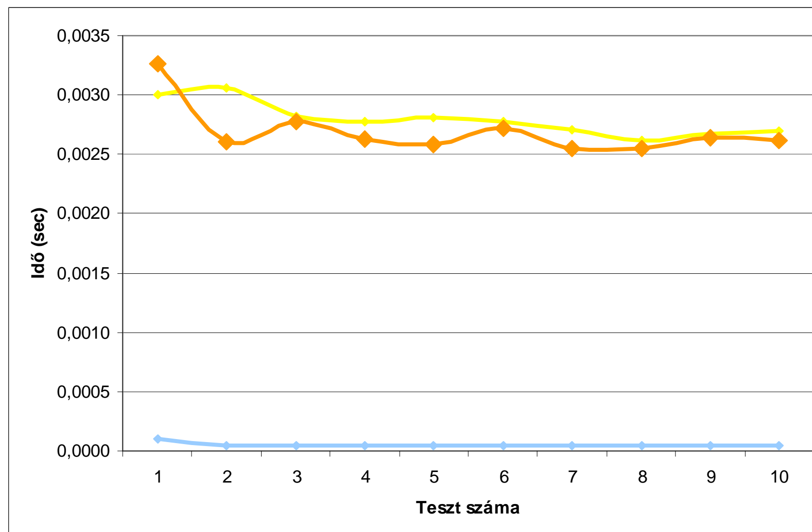
3.4. Az állomány és az adatbázis alapú rendszerek összehasonlítása

Az előző oldalakon először egy állomány alapú, majd egy adatbázis alapú tesztrendszerrel ismerkedtünk meg. Érdemes összehasonlítani az egyes tesztrendszerek futási idejét: a tíz véletlenszerűen kiválasztott tartalom letöltési számának meghatározásához szükséges időt (első teszt), illetve az új tranzakciók elkönyveléséhez szükséges időket (második teszt).

Az első teszt eredményeit a 3.7. ábra, a második tesztét pedig a következő oldalon található 3.8. ábra tartalmazza. Kék színnel az állomány alapú rendszer, sárgás színekkel pedig az adatbázis alapú rendszer eredményei láthatóak. Az időeredmények összehasonlításának célja annak bemutatása, hogy az egyes tesztrendszerek az előnyeik (gyors futás egy adott feladatnál) mellett milyen hátrányokkal bírnak (más feladatok lassabb elvégzése).



3.7. ábra: Tíz véletlenszerűen kiválasztott tartalom letöltési számának meghatározása



3.8. ábra: Tíz új tranzakció elkönyvelése

A fenti táblázatokon jól látszódik, hogy milyen különbségek vannak a kétféle rendszer között. Míg az állomány alapú rendszer nagyon jól szerepel (két nagyságrenddel gyorsabb) az új tranzakciók elkönyvelésében, addig az adatbázis alapú rendszer a véletlenszerű tartalmak keresésével végez sokkal gyorsabban (három nagyságrenddel kisebb a futási ideje). Megállapítható tehát, hogy egyik rendszer sem ideális az összes lehetséges tesztesetnél. Szükség van egy olyan megoldásra, ami mind a kettőnél jobb eredményeket érne el.

Ez hatványozottan igaz az elszámolás konkrét folyamatára is, ami egy elszámolórendszer esetén a leginkább időigényes feladat. Az új megoldásnak az elszámolás idejét is jelentősen csökkentenie kell.

A futási időket nézve logikus ötlet lenne egy adatbázis alapú elszámolórendszert fejleszteni, amit aztán haladó és professzionális technológiákkal ([7], [9]) gyorsabbá, hatékonyabbá, megbízhatóbbá lehetne tenni. Azonban ezek a technológiák sem alkalmasak arra, hogy alapjaiban megváltoztassák a rendszer teljesítményét – például nagyságrendileg kisebb futási eredményt produkáljanak.

Ahogy bemutattam, az állomány alapú rendszer esetén – ugyanúgy, mint az adatbázis alapú rendszernél – az elszámolás során felmerülhet a többszörös joinok problémája. Ha egy lekérdezés során több táblából is származnak adatok, és sok ilyen lekérdezésünk van, akkor a rendszer futása a lezárások (ún. lock)

miatt nagymértékben csökkenhet, esetenként akár le is állhat. Ennek oka az adatbázis-kezelő rendszerek működésében keresendő. Amikor egy táblán egy lekérdezés valamilyen műveletet végez, a művelet típusától függően lock kerülhet az adott táblára. Ez pedig mindaddig fennáll, míg a művelet véget nem ér, addig pedig a tábla más lekérdezésekhez elérhetetlen.

Az, hogy mindkét rendszer a tesztek egy részén rosszul szerepelt, magyarázható azzal is, hogy az adatstruktúra egyik esetben sem ideális szerkezetű az elszámoláshoz. Semelyik megoldásnál nem rendeződnek a tranzakciók úgy, ahogy az az elszámolás készítésekor optimális lenne.

Ezen okok miatt úgy döntöttem, hogy egy teljesen új technológiájú elszámolórendszer fejlesztésébe fogok, amely se nem állomány alapú, se nem adatbázis alapú. Egyetlen lehetőséget látok arra, hogy jóval gyorsabb és hatékonyabb elszámolást lehessen végezni: memória alapú elszámolórendszert kell fejleszteni. Ebben az esetben az elszámolórendszer középpontjában egy memóriában tárolt struktúra áll, ami a régi elszámolórendszerek sebességét jócskán felülmúló megoldást ígér.

4. Puha valós idejű elszámolórendszer tervezése

Az előző fejezet végén elmondottak szerint egy memória alapú elszámolórendszer fejlesztésébe kezdek. Ennek előnye elsősorban gyorsaságában rejlik, mivel egy memória írási vagy olvasási idő nagyságrendileg gyorsabb, mint bármelyik merevlemez-művelet. Ezzel pedig nem csak az állomány alapú rendszereket zárhatom ki, hanem az adatbázis alapúakat is, hiszen az adatbázis mögött is minden esetben valamilyen állomány rejlik, ami magát az adatbázist tartalmazza.

Az új elszámolórendszer ún. puha valós idejű lesz: az egyes tranzakciók fogadásánál a tranzakciók azonnali elkönyvelésére törekszem, azonban a rendszer működése során megengedhetőek csúszások is (pl. hiba esetén, vagy ha egyszerre sok tranzakció érkezik be). Ez nem okoz problémát a rendszer egészének működése során, az elsődleges cél ugyanis az elszámolás a korábbiaknál sokkal gyorsabb és hatékonyabb megvalósítása.

4.1. Elvárások

A létező bevételmegosztásos elszámolórendszerek problémáinak megismerése után immáron elkezdhető az új rendszer tervezése. Egy ilyen jellegű rendszer előállítása előtt fontos, hogy megfelelő (a régi rendszerhez képest jelentős előrelépést jelentő, ugyanakkor még gazdaságosan megvalósítható) elvárásokat tűzzünk ki a rendszerrel szemben.

Gyorsaság:

- A rendszer teljes elszámolásának futási ideje legfeljebb az eredeti rendszer futási idejének 50%-a legyen.

Megbízhatóság:

- A rendszer rendelkezésre állása legalább 99,9%-os legyen.
- A rendszer legyen képes a gyakran és az esetlegesen előforduló hibák kezelésére.
- A rendszer nagyszámú tranzakció fogadására legyen képes.

Rugalmasság:

- A rendszer nyújtson nagyfokú szabadságot a felhasználónak a különböző beállításokkal kapcsolatban.

Biztonság:

- A rendszer legyen ellenálló a különböző külső támadásokkal szemben.
- A rendszer legyen hibatűrő az esetleges belső hibákkal szemben.

Erőforrásigény:

- A rendszer erőforrásigénye ne legyen nagyobb, mint az eredeti rendszer erőforrásigénye.

4.2. A rendszer követelményei

Minden új fejlesztésnél előre szükséges meghatározni azt, hogy az adott feladatoknak való megfeleléshez milyen erőforrásokat igényel a rendszer. Ez elengedhetetlen ahhoz, hogy egy gazdaságosan üzemeltethető programot készíthessünk. Nincs ez másképp jelen elszámolórendszer tervezésekor sem. A következőkben ismertetem, hogy milyen hardveres és szoftveres környezetet igényel az általam készített rendszer.

4.2.1. Hardverkövetelmények

Az elvárásoknál meghatároztam, hogy az elszámolórendszer erőforrásigénye nem lehet nagyobb, mint a jelenleg működő rendszer erőforrásigénye. Tehát a működés nem azért lesz gyorsabb, mert egy jóval erősebb hardveren futtatjuk a rendszert, hanem azért, mert a korábbi technológia helyett egy új, sokkal hatékonyabbat fogok használni. Ennek megfelelően az újonnan fejlesztendő elszámolórendszer hardveres követelményei a tesztrendszer esetén megegyeznek a futtatásához szükséges szoftverek ajánlott erőforrásigényeivel.

4.2.2. Szoftverkövetelmények

Az elszámolórendszer futásához PHP-környezetre van szükség. Semmilyen megkötés nincs a rendszert futtató platform operációs rendszerére, illetve egyéb szoftvereire. Ajánlott valamelyik Linux-disztribúció használata, hiszen megbízhatóság szempontjából ezen operációs rendszerek jobban teljesítenek, mint más megoldások. Szintén ajánlott az Apache webszerver használata, továbbá egy adatbázis-kezelő rendszer megléte. Ez utóbbi egyrészt biztonsági okokból javallott, másrészt az elszámolórendszer működését segítheti.

4.3. A rendszer funkciói

Az elszámolórendszer feladatait két fő részre csoportosíthatjuk: egyrészt tranzakciókkal kapcsolatos feladatokat kell ellátnia (tranzakciók fogadása és könyvelése, kötegelt betöltés és kimentés támogatása, biztonsági mentés készítése), másrészt pedig az elszámolási időszak végén a rendszerben levő tranzakciók alapján egyéb adatok felhasználásával elszámolásokat kell előállítania.

Fontos a feladatokat elkülönítve kezelni, mert van egy alapvető különbség a két feladat között. Míg a tranzakciók fogadása egy folyamatos feladat, amelyet percenként többször is igénybe vehetünk, addig az elszámolás csak havi

rendszerességgel elvégzendő feladat. Így a tranzakciókezelés egy permanensen futó programmodul, amely felhasználói interakciót nem igényel. Az elszámolás ennek pontosan az ellentéte: felhasználói beavatkozás hatására kerül sor a futtatására.

4.3.1. Tranzakciók fogadása

A rendszer legtöbbet használandó funkciója a tranzakciók kezelése lesz. Amikor egy tranzakció beérkezik, azaz egy tartalom letöltése megtörténik, akkor erről valamilyen módon értesíteni kell az elszámolórendszert. Hogy ez az értesítés milyen formában történik, a tervezés jelenlegi fázisában még nem eldöntött, azonban egy koncepcionális kérdést mindenképpen el kell dönteni: a tranzakcióról a rendszer aktív vagy passzív módon értesül-e. Az aktív mód ebben az esetben azt jelöli, hogy a rendszer (megszabott időközönként) aktívan fogadja a tranzakciókat (pl. lekérdezi egy adatbázisból, kiolvassa egy állományból). Passzív mód alatt pedig azt értem, hogy a rendszer tranzakció érkezésekor valamilyen figyelmeztetés hatására kéri le a tranzakciókat, esetleg a figyelmeztetéssel együtt kapja meg azokat. Előbbi eljárás mellett döntöttem, mert több érv szól mellette, mint a passzív mód mellett. Aktív fogadásnál sokkal könnyebben meghatározható a rendszer viselkedése, kizárható a kockázata annak, hogy a rendszer a túl sok tranzakció hatására lefagyjon, vagy működésében egyéb zavar álljon be. További előny, hogy ennél a megoldásnál a rendszer működése is sokkal inkább determinisztikus, hiszen előre beállított paraméterek függvényében határozhatjuk meg a rendszer tevékenységét.

4.3.2. Betöltés

A rendszernek mindenképpen támogatnia kell valamilyen módon azt, hogy egyszerre sok tranzakció egyszerűen, tömbösítve bevitelre kerüljön. Erre két okból van szükség: egyrészt valamilyen rendszerhiba utáni felállás alkalmával, amikor egy korábbi memóriamentést akarunk betölteni, másrészt ugyanígy

szükséges a betöltés az elszámoláskor. A betöltésre két lehetőséget látok: történhet adatbázisból, esetleg állományból. Ez utóbbi megvalósítás mellett szól az, hogy így nincs szükség külön adatbázisra a rendszer mellé, és egy ilyen megvalósítás sokkal egyszerűbb és megbízhatóbb, mint egy adatbázis.

A betöltés során a betöltendő állomány feldolgozása soronként történik. Ennek gyors és hatékony megvalósítása fontos feladat, mert a rendszer működését alapvetően befolyásolhatja az esetleg előforduló lassú memóriatöltés. Ez a probléma elsősorban a kimentésnél jelentkezhet, azonban már a betöltésnél is figyelembe kell venni.

4.3.3. Kimentés

A kimentés a betöltés funkció párjaként működik. Az előző döntés értelmében kimentéskor a memória tartalma egy szöveges állományba íródik. Érdemes elgondolkodni azon, hogy mennyire legyen ez a szöveges adatbázis szabványszerű. Úgy gondolom, hogy tesztelési, ellenőrzési céllal esetleg szükség lehet a memóriamentés egyéb adatbázisba való betöltésére, így a legjobb megoldás az, ha valamely elterjedt és egyszerű adatbázis formátumát használom. Ilyen például a MySQL-adatbázis, így a memóriát SQL-kiterjesztésű, MySQL-szabványnak megfelelő formában fogom kimenteni. (Így természetesen a betöltés is egy ilyen állományból történik.)

Mivel a kimentés során egy állományt készít a rendszer, már most fontos odafigyelni a biztonságos és gyors működés közötti egyensúlyra. Minél több memóriamentést készítünk, annál nagyobb a biztonság, viszont jelentős overheadet okozhatunk a rendszernek. A túl ritka memóriamentés sem jó megoldás, hiszen szükség esetén (pl. áramszünet) a rendszer felállása (ahol felállás alatt az újonnan beérkező tranzakciók mielőbbi valós idejű elkönyvelését értem) hosszú ideig is eltarthat, ha nincs friss mentés. A rendszer működése szempontjából tehát fontos a kimentési időközöket úgy megválasztani, hogy a működés biztonságos legyen, de az overhead se legyen túl nagy.

4.3.4. Elszámolás

Egy elszámolórendszernél – ahogy a neve is mutatja – a legfontosabb funkció természetesen az elszámolás maga. Az elszámolás egy felhasználói beavatkozást igénylő funkció lesz (ez általában így történik a valós elszámolások során is). Az elszámolás és az elszámolórendszer futása ezért egymástól elkülönítve történik: miközben a rendszer továbbra is működik, és fogadja a beérkező tranzakciókat, addig az elszámolómodul attól elkülönítve, felhasználói beavatkozás hatására aktivizálódik. Az elszámolás során az adott elszámolási időszak (a legutóbb lezárt hónap) teljes mentését töltjük be, és azon futtatjuk le az elszámoláshoz szükséges algoritmusokat, szűréseket.

Az elszámolásnál az elsődleges szempont a gyorsaság. Mivel az elszámolás folyamatánál előfordulnak adatbázis- és állományműveletek, fontos, hogy ezek ne befolyásolják a futási időt drasztikusan, azaz ne veszítsük el a memória alapú működés okozta helyzeti előnyt. A megvalósításnál mindvégig ezt fogom szem előtt tartani.

4.4. A rendszer működése

Az elszámolórendszer említett funkciói meghatározzák, hogy milyen alapfeladatokat kell ellátni, azonban semmit nem mondanak a rendszer belső működéséről. A fejlesztés során a belső feladatok ellátására külön-külön függvényeket fogok írni, így téve eleget a struktúrált programozás kívánalmainak.

Az elszámolórendszer legfontosabb feladatai a beérkező tranzakciók fogadása, eltárolása és végül maga az elszámolás. Ezért aztán alapvető fontosságú annak a struktúrának a meghatározása, amelyben a tranzakciók adatait tároljuk. Mint ahogyan azt a 3. fejezetben bemutattam, az állomány alapú és az adatbázis alapú elszámolórendszerek lassúsága, illetve optimalizálatlansága elsősorban abból adódik, hogy az adatokat nem megfelelő struktúrában tárolják. Egy elszámolás során esetenként akkora mennyiségű, és olyan speciális rendezőelv alapján szűrt adatokat kell használni, amik szükségessé teszik egy erre

a célra tervezett adatstruktúra használatát. Sem az állomány alapú rendszernél a chargelog, sem pedig az adatbázis alapú rendszernél a tranzakciókat számláló táblázat nem ideális, amit jól mutatnak a mérési eredmények is.

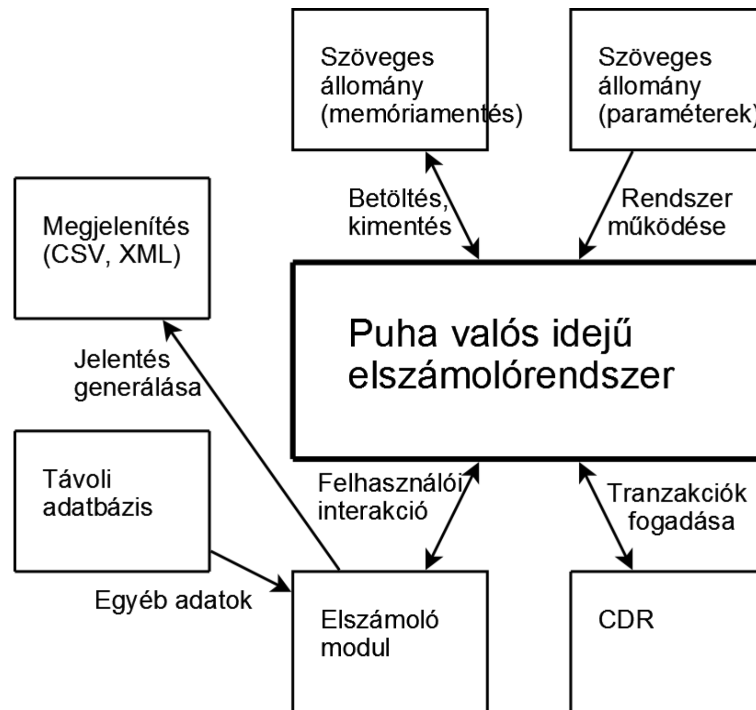
Ahhoz tehát, hogy az új elszámolórendszer gyorsan és hatékonyan működhessen, meg kell találni az elszámoláshoz leginkább illő adatstruktúrát. Ez pedig nem lehet más, mint egy többszintű fa, amelyet az elszámoláshoz optimalizálva építünk fel. A fa az az adatstruktúra, aminél az adatok elrendezése biztosítékot nyújt arra, hogy az elszámolásnál az egyes tulajdonságok, tulajdonságcsoportok alapján hasonló tranzakciók egymás után következzenek, ezzel növelve a futás gyorsaságát.

Definíció: a fa egy olyan (esetünkben irányítatlan) gráf, amelynek bármely két csúcsát pontosan egy út köti össze. Másképp megfogalmazva az összefüggő körmentes gráfokat fának nevezzük. [12]

Konkrét megvalósítás: e többszintű fát programozási szempontból tökéletesen helyettesít egy asszociatív tömb. A fa gyökeréből a levelek felé csúcspontról-csúcspontra egy-egy tartalmat, illetve a tartalom tulajdonságait járjuk be, míg a levél a pillanatnyi letöltésszámot tartalmazza.

E megoldás előnyei mind a könyvelésnél, mind az elszámolásnál mutatkoznak. Előbbi esetben jelentős eredmény, hogy egy már könyvelt tartalom, illetve a most először fogadott tartalom nem kell, hogy elváljanak egymástól. Elegendő csupán megcímezni a fát (tömböt) a tartalom tulajdonságai alapján, és a levelet inkrementálni. Amennyiben még nem volt ilyen tartalom, az inkrementálás hatására a levél egyes értéket vesz fel, ha pedig volt már ilyen, akkor is megfelelő a fenti módszer, és a letöltésszám eggyel nő. Elszámolásnál az biztosít előrelépést, hogy a tartalmak a struktúrának köszönhetően nem véletlenszerűen helyezkednek el egymáshoz képest, hanem a tartalom tulajdonságai alapján csoportosítva, azaz a fa „rendes" (pl. ABC-sorrend szerinti) bejárásával lehetőség nyílik az elszámoláshoz megfelelő sorrendben hozzájutni a tartalom–letöltések száma párosokhoz.

A rendszer működésének összefoglalását a következő oldalon található 4.1. ábra mutatja. (Csak a főbb részeket és kapcsolatokat tüntettem fel.)



4.1. ábra: Az elszámolórendszer működése

4.4.1. Alapvető függvények

A rendszer fejlesztése előtt érdemes átgondolni, hogy milyen alapvető függvények szükségesek a különböző funkciók ellátásához. Ezen függvényeket több csoportba lehet osztani aszerint, hogy milyen feladatok ellátására készülnek.

Ki- és bemeneti adatokkal kapcsolatos függvények: mindenképpen szükség lesz a memóriát szöveges állományba mentő, illetve szöveges állományból a memóriát feltöltő függvényekre. Ide kapcsolódnak a biztonsági mentések kezelését végző függvények is. Ezek megvalósítását már a fejlesztés elején el kell végezni.

Rendszerbeállítási függvények: a rendszer működését nagymértékben befolyásolhatják a különböző konfigurációs állományokban megadott paraméterek. Ezenfelül pedig olyan adatok megadására is szükség van, amik a rendszer működéséhez elengedhetetlenek. A fejlesztés elején szükséges kifejleszteni az ezen funkcióval összefüggő függvényeket.

Tömb- és faátalakító függvények: a tranzakciók fogadása során a rendszer egy többszintű fát épít a memóriában. Ez egy ideális adatstruktúra az elszámolás

elkészítéséhez, azonban szükség lesz olyan függvényekre, amik a fát átalakítják más fává vagy más adatstruktúrává, amivel az elszámolás konkrét lépései megvalósíthatóak. Már a fejlesztés elején el kell készíteni azokat a függvényeket, amik a fa és az asszociációs vagy asszociáció nélküli tömbök közötti átalakításokat végzik.

Tranzakciókezelő függvények: szintén a fejlesztés első szakaszában szükséges a tranzakciók fogadását (a tranzakciók kérését és elkönyvelését) megvalósító függvények fejlesztése. Ezek egyrészt a távoli tranzakciófogadóval folytatnak kétirányú kommunikációt (az ún. CDR segítségével fogadhatóak a beérkező tranzakciók, erről bővebben az 5.7.1. részben), másrészt pedig a megkapott tranzakciókat építik bele a memóriában tárolt többszintű fába.

Elszámolással kapcsolatos függvények: az elszámolás tulajdonképpen nem más, mint bizonyos szempontok alapján csoportosított tranzakciók összeszámolása, és egyéb adatok begyűjtése (például egy távoli adatbázisból). A fejlesztés középső szakaszában a fő feladat az ezzel kapcsolatos függvények elkészítése, amelyek részint a felhasználóval folytatott kétirányú kommunikációt hivatottak megvalósítani, másrészt pedig az összegzési feladatokat látják el.

Megjelenítéssel kapcsolatos függvények: az elszámolás az egyetlen olyan rendszerfunkció, amely során a felhasználóknak szánt kimenetet állít elő a rendszer. A jelenlegi tervek szerint a rendszer több kimeneti formátumot is támogatni fog, elsősorban a CSV-t, másodsorban pedig az XLS-t. (A későbbiekben további kimeneti formátumok támogatása is elképzelhető.) A megjelenítő-függvényekre a fejlesztés végén kerül sor.

Egyéb függvények: szükséges egy hibakezelő függvény, ami a rendszer nem várt működését kezeli, naplózza. Mivel a rendszerben több bonyolultabb, összetett függvény is lesz, ezek prognosztizálják segédfüggvények alkalmazását is. A segédfüggvények fejlesztésével áttekinthetőbb struktúra hozható létre, így az ilyen esetekben ezt a módszert fogom alkalmazni.

5. Bevételmegosztásos elszámolórendszer fejlesztése

Az új, puha valós idejű elszámolórendszer tervezése után megkezdődhet a rendszer fejlesztése. Az alábbiakban a fejlesztést fő részei szerinti bontásban mutatom be. Először a fejlesztői környezetet ismertetem, ezután a rendszer különböző funkcióit megvalósító függvényeket és segédfüggvényeket részletezem. A rendszer működésének részletes bemutatása után a tartalomkezelő rendszerhez való kapcsolódást ismertetem, végül pedig biztonsági és egyéb kérdésekkel fogok foglalkozni.

5.1. Fejlesztői környezet bemutatása

A fejlesztőkörnyezetet a rendszerkövetelmények meghatározásához hasonlóan hardveres és szoftveres részre osztva mutatom be.

5.1.1. Hardverkörnyezet

Hardveregység	Leírás
Alaplap	Compaq gyártmányú, Socket478
Processzor	Intel Pentium IV, 1700 MHz
Memória	256 MB DDR RAM
Merevlemez	40 GB

5.1. táblázat: A tesztrendszer hardveregységei

A tesztrendszer főbb paramétereit az 5.1. táblázat tartalmazza. Az elszámolórendszer fejlesztését egy x86 architektúrájú számítógépen végzem. A program nem tartalmaz hardverspecifikus összetevőket, így a futtatás tetszőleges

architektúrán lehetséges, ami eleget tesz a szoftverkörnyezet bemutatásánál leírt feltételeknek.

5.1.2. Szoftverkörnyezet

Szoftveregység	Verzió	Előnyök
Linux (Kubuntu)	6.10 (Edgy)	Ingyenes, megbízható
Webszerver (Apache)	2.0.55	Ingyenes, elterjedt
PHP	5.1.6	Ingyenes, megbízható, elterjedt
Adatbázis (MySQL)	5.0.24a	Ingyenes, megbízható, elterjedt

5.2. táblázat: A tesztrendszer szoftveregységei

A tesztrendszer konkrét leírását (a választott komponensek előnyeivel) az 5.2. táblázat tartalmazza. Az elszámolórendszer fejlesztését Apache webszerveren, PHP-„nyelven” és MySQL-adatbázis segítségével végzem. A tesztrendszeren Linux operációs rendszer fut, a fejlesztéshez szükséges műveletek pedig SSH-protokollon keresztül folynak.

Egy rendszer fejlesztésénél fontos szempont az, hogy lehetőleg minél inkább platformfüggetlen legyen. Ez ezúttal sincs másképp. A fejlesztés során végig szem előtt tartom azt, hogy operációs rendszertől, webszervertől és adatbázistól független rendszert készítsek. Az egyetlen szükséges kritérium a PHP-környezet megléte, azonban a rendszer lefelé is kompatibilis (nincsenek benne PHP5-specifikus részek).

A PHP-nyelvet az esetek döntő többségében webes alkalmazásoknál használják. A legtöbb dinamikus honlap – részben vagy egészben – PHP-vel generált HTML-kód alapján épül fel. Az elszámolórendszernél a tranzakciók fogadásánál, elkönyvelésénél és a különböző I/O-műveleteknél nincs szükség semmilyen grafikus megjelenítésre, sem pedig felhasználói beavatkozásra. Ezért úgy döntöttem, hogy a rendszert parancssorból fogom futtatni (az ilyen módon futtatott programok összefoglaló neve shell szkript). Ez egyrészt még gyorsabbá teszi a különféle műveletek elvégzését, másrészt számos olyan „mellékhatással” jár, amik segítik a program futását. Ezek közül egy – ami talán a legfontosabb – az, hogy PHP parancssori futtatásánál a max_execution_time paraméter 0-s

értéket kap, ami definíció szerint korlátlan futási időt garantál. Erre azért van szükség, mert míg a különböző webes alkalmazásoknál a futási idők általában másodperces nagyságrendűek, addig a shell szkripteknél ez az idő sokkal nagyobb (órák, napok, akár végtelen futás is szükséges lehet). A parancssori futtatást csak a végleges program futtatása során fogom használni, a fejlesztés folyamatában a rendszert webes alkalmazásként használom. Ekkor a megfelelő maximális futási időt a PHP tulajdonságainál szükséges beállítani. [1]

A rendszer működéséhez az alábbi modulok lefordítása szükséges (az alapértelmezett könyvtárakon kívül):

- iconv (különböző kódolású szövegek kezelésére szolgál);
- mysql (a PHP MySQL-adatbázist támogató könyvtára);
- ssl (a HTTPS kommunikációhoz szükséges).

5.2. Belső függvények

Az elszámolórendszer fejlesztését modulárisan végzem, azaz minden külön funkciót egy-egy függvény valósít meg. Egy feladatot tehát kisebb, egyszerűbb részfeladatokra bontok, és ezen részfeladatok segítségével oldom meg az eredeti feladatot. A fő feladatok jelen esetben a tranzakciók kezelése, illetve az elszámolás elkészítése. Ezeket részfeladatokra bontom, amik egy-egy függvénynek felelnek meg. Esetenként pedig ezeket a részfeladatokat is tovább bontom, a függvények további függvények, segédfüggvények hívásával valósítják meg az eredeti funkciójukat. A moduláris programozás egyrészt segít a program átláthatóbbá tételében, másrészt egyszerűsíti a tesztelést, harmadrészt segítséget nyújt a későbbi továbbfejlesztésekben. (A moduláris programozás elveiről bővebben: [13].)

5.2.1. Tömb- és faátalakító függvények

Az elszámolórendszer működésének középpontjában az a többszintű fa áll, amely optimális adatszerkezetet biztosít a beérkező tranzakciók fogadásához és tárolásához. Azonban néhány funkció ellátáshoz szükség van a többszintű fától eltérő szerkezetű adatstruktúrákra is. Az ezen feladatot ellátó függvényeket mutatom be a következőkben.

- flatminator(): a függvény feladata a többszintű fa mélységi bejárása, és a levelekig vezető, legalább egy élben különböző utak eltárolása. A flatminator() rekurzív működésű, a meghívásánál két bemeneti paramétere van: az első egy többszintű fa, a második pedig az idáig vezető út. Visszatérési értéke nincs, egy globális változóba írja a különböző utakat.
- list_unique(): a függvény működése bonyolult lenne, ha segédfüggvényei (querior(), uniqueor(), és részben a flatminator()) helyett mindent ebben kellene implementálni. Így működése jóval egyszerűbb: a querior() által előállított, uniqueor() segítségével felépített és a flatminator()-ral kiterített fát dolgozza fel, és visszaadja az eredményeket a bemeneti paraméterének megfelelő sorrendben.
- querior(): a függvény egy attribútumlistát kap. A kapott attribútumok kétfélék lehetnek: vagy egy adott konkrét érték szerepel az attribútumnál, vagy pedig csak maga az attribútum. A querior() az adatstruktúra alapján készít egy lekérdezést: ha a struktúra soron következő eleme szerepel az attribútumlistában, akkor az – attól függően, hogy szerepel-e érték feltételként – bekerül a lekérdezésbe, feltétellel vagy anélkül. Ha az elem nem szerepel a kapott listában, akkor a lekérdezés összeállításánál nem vesszük figyelembe.
- to_array(): a függvény a struktúrát bejárva, annak megfelelően épít fel egy nem asszociatív tömböt az általa kapott asszociatív tömbből.

A to_array() függvényt több más függvény is használja a különféle tömbökkel kapcsolatos műveleteik során.

- to_assoc(): a függvény a struktúra alapján épít asszociatív tömböt a kapott nem asszociatív tömb segítségével. A to_assoc() függvényt több más függvény is használja a különféle tömbökkel kapcsolatos műveleteik során.
- uniqueor(): ez a rekurzív függvény a querior() által előállított lekérdezés segítségével az eredeti többszintű fát járja be, és egy globális változóban tárolt másik fát épít fel belőle. A lekérdezés azt határozza meg, hogy az eredeti fa egy adott ágát bejárjuk-e. A függvény a bejárás végén az új fa leveleit is kitölti a régi fa levelei alapján. A rekurzióhoz szükséges egy temporális változó is, amiben eltárolásra kerül az eddig bejárt útvonal. Az uniqueor() függvény nagy fontosságú, az elszámolásnál is kulcsszerepet kap, hiszen maga az elszámolás nem más, mint az eredeti fa átalakítása egy olyan fává, amiben a nem fontos szintek elhagyásra kerülnek, a levelek pedig jól összegzett értékeket tartalmaznak.

5.2.2. Tranzakciókezelő függvények

A rendszer két fő feladata közül az egyik a tranzakciók folyamatos fogadása és könyvelése. A tranzakciókat egy másik rendszerből, egy távoli adatbázisból kérjük le, ahova azok a vásárlás pillanatában érkeznek be. Ebben a távoli adatbázisban egymás után sorakoznak a beérkező tranzakciók, mindegyik egy egyedi azonosítószámmal van ellátva (erre – többek között – az elszámolórendszer hiba utáni újraindulásakor és az általános tranzakció-lekéréseknél is szükség lesz). A következőkben a tranzakciókezeléshez szorosan kapcsolódó függvényeket mutatom be.

- flush_buffer(): a függvény fő feladata a távoli adatbázissal való kommunikáció (az új tranzakciók lekérése), majd a beérkezett tranzakciók könyvelése. Bemeneti paraméterként a legutóbb

elkönyvelt tranzakció sorszámát, illetve az ehhez tartozó év-hónap párost kapja meg. A távoli adatbázist a flush_buffer() közvetve éri el, a távoli kiszolgálón található adatbázis-kezelő függvény segítségével, amelyet GET-paraméterek segítségével vezérel (elküldi a struktúrát és az utolsó elkönyvelt tranzakció azonosítószámát). A távoli rendszer visszatérési értéke az újonnan beérkező tranzakciókból felépített tömb, amit a kommunikációs csatornán való áthaladás előtt szerializált (serialize() függvény), a flush_buffer() tehát a kapott adatfolyamot visszaalakítja (unserialize() függvény). Az új tranzakciókat egyenként kezeli a függvény, minden esetben vizsgálja, hogy az adott tranzakció az aktív hónaphoz tartozik-e, ha igen, akkor elkönyveli (to_array() függvénnyel átalakítja a tranzakciót, majd az incrementor() függvénnyel a fába helyezi), ha nem, akkor egy globális változóval jelzi a hónapváltást. A függvény visszatérési értéke minden esetben a legutóbb elkönyvelt tranzakció száma, és a hozzá tartozó év – hónap páros.

- id_load(): a függvény feladata a legutóbb elkönyvelt tranzakció azonosítószámának kiolvasása. A nagyobb biztonság érdekében ezt a számot nem a memóriában tároljuk, nem is egy adatbázisban, hanem egy szöveges állomány egyetlen tartalmaként.
- id_save(): a függvény az id_load() párjaként a legutóbb elkönyvelt tranzakció azonosítószámát írja bele egy szöveges állományba.
- id_to_month(): a függvény feladata a bemeneti paraméterként kapott tranzakció-azonosítószámhoz tartozó év – hónap páros lekérdezése a távoli kiszolgálótól. Erre üres indulásnál, illetve hiba utáni rendszer-helyreállítás esetén van szükség. Visszatérési értéke tehát a tranzakcióhoz tartozó év – hónap páros.
- inc_month(): ez a segédfüggvény a hónapfordulók esetén használatos. Csak annyi feladata van, hogy az általa kapott év – hónap párost követő hónap év-hónap párosát visszaadja. A függvény

figyel az évfordulásra, illetve arra, hogy a hónap mindig két karakteren kerüljön eltárolásra.

- incrementor(): a függvény feladata a többszintű fa leveleibe való írás. Egyrészt a betöltésnél van erre szükség, mikor a memóriamentést tartalmazó tömböt újra felépítjük, másrészt pedig a tranzakciók fogadásánál használjuk. Amennyiben egy konkrét értékkel hívjuk meg, akkor a fa megadott levelének megfelelő helyre az értékkel megnövelt szám kerül. Ha az adott levél még nem létezett, akkor a számot írjuk oda. Ha érték nélkül hívjuk meg a függvényt, akkor az alapértelmezett egyes számmal inkrementáljuk a megfelelő levelet. Az incrementor() rekurzív működésű, a kapott fán az útvonalnak megfelelően végighalad, majd az értékkel növeli a levél tartalmát. (Az incrementor() kódját a Függelékben csatoltam.)
- month_to_id(): az id_to_month() függvényhez hasonló működésű függvény is a távoli kiszolgálóval kommunikál. A feladata a kapott év – hónap pároshoz tartozó legkisebb tranzakció-azonosítószám visszaadása.

5.2.3. Ki- és bemeneti adatokkal kapcsolatos függvények

A ki- és bemeneti adatokkal kapcsolatos függvényeket akkor használjuk, ha a memóriában tárolt adatokat kívánjuk szöveges állományba menteni, illetve egy szöveges állomány tartalmát szeretnénk a memóriában újrafelépíteni. Elsősorban a PHP állománykezelő függvényeit (fopen(), fgets(), fputs() stb.) használom, néhány esetben ettől eltérő beépített függvények is alkalmazásra kerülnek. Az előzőekben bemutatott id_load() és id_save() függvények is állománykezelési feladatokat látnak el, ám logikai szempontból inkább a tranzakciókezeléshez tartoznak.

- attrib_load(): ez a segédfüggvény akkor hasznos, ha egy memóriamentést tartalmazó állományból szeretnénk egy adott attribútumhoz tartozó egyedi előfordulásokat kikeresni. Ilyenkor

nincs szükség a teljes fa memóriában való felépítésére és erőforrásigényes (memória, de legfőképpen idő) keresések folytatására, elegendő az állományt soronként végigolvasni, és a különböző attribútumértékeket feljegyezni. A függvény tehát a bemeneti attribútum és állomány alapján adja vissza az egyedi előfordulásokat.

- count_files(): a függvény feladata egy adott könyvtárban levő állományok megszámolása. A memóriamentések számolására használom.
- del_old(): a függvényt akkor használjuk, amikor egy vagy több memóriamentést kívánunk törölni, ha a számuk elérte a konfigurációs állományban megadott maximális számot. A törlés mindig a legrégebbi állományt érinti, és mindaddig folyik, amíg legfeljebb annyi memóriamentés lesz a könyvtárban, amennyit a felhasználó engedélyezett.
- load_last(): a függvény feladata a rendszer futásának elején a legutóbbi memóriamentés megkeresése, majd betöltése. A legutóbbi memóriamentés az, ami legkésőbb készült, így feltételezésünk szerint a legtöbb már elkönyvelt tranzakciót tartalmazza.
- sql_load(): a legfontosabb függvény a rendszer biztonságos működése szempontjából. Feladata egy paraméterként megkapott állomány beolvasása, és tartalma alapján a memóriában a többszintű fa felépítése. Ezen feladatok elvégzéséhez rendelkezésére áll az adatstruktúra, ami alapján – felhasználva a memóriamentés elején található, SQL-szabványos adatbázis-leíró részt – a beolvasást és faépítést végzi. A szöveges állományt soronként értelmezi, felismeri a deklarációs részt és az adatokat tartalmazó sorokat, közben pedig figyelmen kívül hagyja a szabványos kommenteket is. A deklarációs rész alapján az sql_load() függvény felismeri, hogy az adatsorokban az egyes értékek hol helyezkednek el. Az adatokat tartalmazó állományrészt is soronként olvassa be, és az incrementor() függvény

segítségével építi fel a többszintű fát a memóriában. Az sql_load() függvényt egyrészt hiba utáni rendszer-helyreállításra használom, másrészt elszámolásnál a teljes havi mentések beolvasását is e függvény segítségével végzem.

- sql_save(): az sql_load() párjaként a rendszer biztonságos működéséhez elengedhetetlen, illetve a hónapforduláskor használom. Az egyetlen opcionális bemenete a kimentési állomány neve, ha nincs megadva, akkor egy időbélyeg generálásával határozza meg azt. Az sql_save() először az állomány deklarációs részét építi fel az adatstruktúra alapján, majd a memóriában levő többszintű fát alakítja át a flatminator() függvény segítségével a levelekhez vezető különböző utakra. Az utakat tartalmazó tömbön soronként végigfut, és az állomány adatsorokat tartalmazó részébe írja a megfelelő tartalom – letöltésszám párosokat szimbolizáló sorokat.

5.2.4. Rendszerbeállítási függvény

A konfigurációs állomány lényege, hogy a felhasználó különböző paraméterek beállításával gyorsan és egyszerűen egy tartalomkezelő rendszerhez illeszthesse az elszámolórendszert. Ezen paraméterek egyrészt a rendszer alapvető információit tartalmazzák (például itt kell megadni az adatstruktúrát, a távoli kiszolgálóhoz való kapcsolódási pontot stb.), másrészt pedig különböző finomhangolásokra is lehetőséget biztosítanak.

- config_load(): a függvény feladata a konfigurációs adatokat tartalmazó állomány beolvasása, és az ott található adatok változókba való beírása. A konfigurációs állomány felépítése előre megszabott, kötelező és opcionális paraméterek megadását teszi lehetővé. A beállítási adatok kétféleképpen kerülnek eltárolásra: az adatstruktúrához tartozó paramétereket (struktúra, letöltésszám) nem asszociatív tömbbe írja a config_load(), míg a többi paraméter asszociatív tömbbe kerül. Ennek oka az, hogy a struktúrában

alapvető fontosságú maga az egyes elemek sorrendje is, így ott nem pusztán az egyes attribútumnevek eltárolása a lényeg, hanem a hozzájuk tartozó szint is (ami a többszintű fa építésekor játszik szerepet). A konfigurációs állományban az adatstruktúrához kapcsolódó paraméterek mellett a memóriamentéshez kapcsolódó értékek állíthatóak (backup). Lehetőség van a kezdő év – hónap páros megadására, ahonnan az elszámolás indul (init), ugyanitt a hibakezelő függvény állománya is megadható. A távoli kiszolgálóval és adatbázissal kapcsolatos beállítások megadása alkotja a paraméterek negyedik fő csoportját (remote_db). Végül pedig a tranzakciók fogadásához és egyéb műveletekhez szükséges adatok is megadhatóak (buffer). A config_load() működése szempontjából a ki- és bemeneti függvények között is lehetne, ám fontossága és szerepe miatt emeltem ki egy külön kategóriába.

5.2.5. Elszámolással kapcsolatos függvények

Az elszámolás során a már lezárt, egész hónapos memóriamentésekből építjük fel a többszintű fát, majd az elszámolások készítéséhez az alábbi függvényeket használjuk. Ezeken kívül még a tömb- és faátalakító függvényekre, illetve a megjelenítési függvényekre van szükség a végleges elszámolások elkészítéséhez.

- aggregator(): a függvény rekurzív módon aggregálja a többszintű fát a számára elkészített aggregálási feltételeknek megfelelően (ezt állítja elő a querior() függvény). Az aggregált eredménytömb egy globális változóba kerül, így az aggregator()-nak nincs kimenete. A rekurzió során a függvény szélességi kereséssel bejárja a fát, működése szintenként attól függ, hogy az adott szintnek megfelelő feltétel éppen mi. Ha a feltétel egy konkrét érték, akkor csak az adott értéknek megfelelő farészt járja be az algoritmus. Ha a feltétel *, akkor minden adott szinten levő érték aggregálásra kerül. Ha a feltétel aktuális értéke {, akkor az egész szint kibontásra kerül. A

rekurzív működéshez eltárolásra kerül az addig bejárt útvonal, amit a függvény minden meghívásánál bemeneti paraméterként megkap (rekurzív hívásnál pedig természetesen elkészíti az új útvonalat, és átadja azt az újonnan meghívott aggregator() függvénypéldánynak). Az aggregator() rekurziója a levelekig tart, ahol is egy incrementor() függvényhívással a levél tartalmát az eredményeket tartalmazó tömbbe íratjuk bele. A függvény működése az egész elszámolás kulcsa, így néhány példán keresztül mutatom be az aggregator() függvényt. (Az aggregator() kódját a Függelékben csatoltam.) Néhány példalekérdezést, illetve az azokra adott eredményeket tartalmazza az 5.3. táblázat.

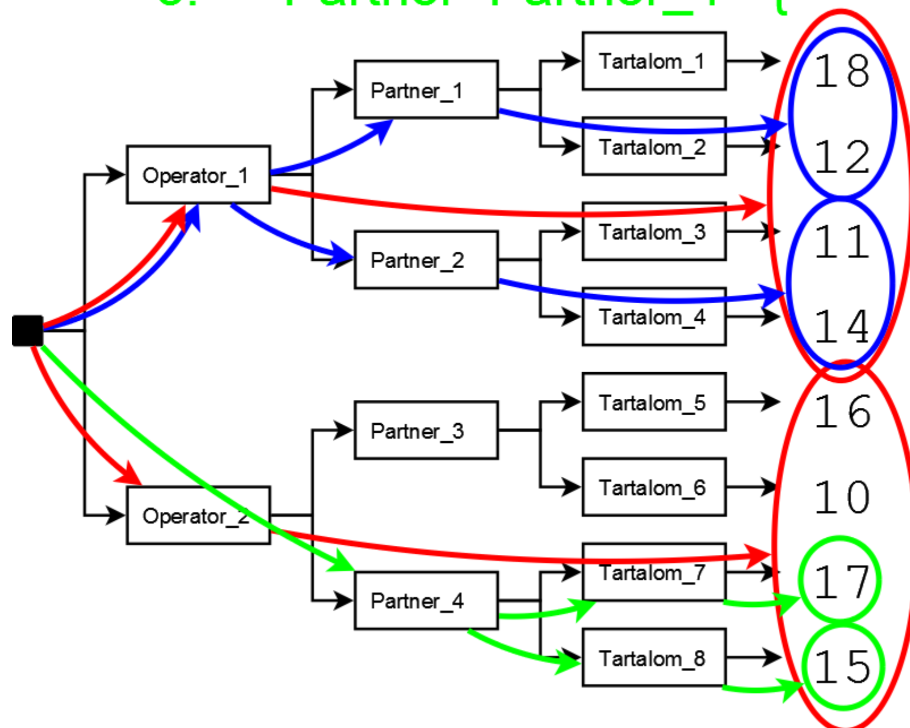
Lekérdezés	Eredmény
{ – * – *	Operator_1 – 55
	Operator_2 – 58
Operator=Operator_1 – { – *	Operator_1 – Partner_1 – 30
	Operator_1 – Partner_2 – 25
* – Partner=Partner_4 – {	Partner_4 – Tartalom_7 – 17
	Partner_4 – Tartalom_8 – 15

5.3. táblázat: Az egyes lekérdezések és az aggregator() függvény hozzájuk tartozó válasza

A függvény feladata tehát egy adott fa bejárása és összegzése a lekérdezések alapján. A kimenet minden esetben egy újabb fa, ami a szükségtelen farészeket elhagyja, és a leveleket a lekérdezéseknek megfelelően összegzi. A fenti táblázat eredményei nem az aggregator() kimenetei, hanem a kimenetek flatminator() függvénnyel kezelve. A különböző színek különböztetik meg a lekérdezéseket, a nyilak a többszintű fa bejárását szimbolizálják, a levelekben eltárolt letöltési számok bekarikázása pedig az összegzés végeredményéhez szükséges összeadásokat mutatják. A fa gyökerét a bal oldali fekete négyzet jelöli, ez felfogható PHP-ben a fa tartalmát befogadó tömb neveként, az egyes szinteken levő nevek pedig a tömb címzéséhez szükségesek. A Tartalom_1 letöltési számához például az alábbi hivatkozással juthatunk el:

```
$tomb [ ”Operator_1” ] [ ”Partner_1” ] [ ”Tartalom_1” ]
```

E változó formátuma például egész szám lesz, és 18-as értéket vesz fel. Az aggregator() függvény működését az 5.1. ábra segítségével szemléltetem.



5.1. ábra: Az aggregator() függvény működése

A fenti egyszerű példa (háromszintű fa, kevés elágazás) jól szemlélteti az aggregator() függvény működését. A fában levő szintek és az egyes szinteken levő elemek számának növelésével a függvény működése semmit sem változik, a bejárás ugyanígy történik meg. (Természetesen egy bonyolultabb struktúrájú fa bejárása és adott lekérdezésnek megfelelő összegzése nagyobb memóriaigényű feladat, és futási ideje is több, mint az egyszerűbb fák esetén. Ugyanígy a flatminator() memóriaigénye és futási ideje is növekedik a végleges eredmények előállításánál.)

- tablazat(): a függvény feladata az elszámolások elkészítéséhez szükséges adatok logikai táblázatba foglalása, amelyet aztán a megjelenítéssel kapcsolatos függvények alakítanak át tényleges elszámolási állományokká.

  A függvény több segédfüggvény gyűjteménye:

  - tablazat_ujtablazat() – új táblázat létrehozása;
  - tablazat_ujsor() – új sor kezdése a logikai táblázatban;
  - tablazat_cella() – a soron következő cellába egy érték írása;
  - tablazat_lezaras() – a táblázat lezárása.

  A segédfüggvények visszatérési értéke új sor létrehozása és a táblázat lezárása esetén a táblázat sorainak száma, új cella létrehozásakor pedig az aktuális cella koordinátái. Új táblázat készítésekor a segédfüggvény kimenete az új táblázatot tartalmazó változó.

Az elszámolás fontossága miatt külön tárgyalom az elszámolás teljes folyamatát az 5.5. részben.

5.2.6. Megjelenítéssel kapcsolatos függvények

Az elszámolás során létrejött eredmények tulajdonképpen mindent tartalmaznak, amire a partnerekkel való elszámoláshoz szükség van, azonban szükséges az adatokat feldolgozható formában elérhetővé tenni. A megjelenítési függvények feladata ezen átalakítások elvégzése, és a felhasználók számára szükséges formátumú állományok előállítása. Mivel az elszámolás után létrejött adathalmaz egyszerűen alakítható át tetszőleges formába, így a későbbiekben más kimeneti formátumok támogatása is könnyen megoldható.

- number2excel(): ez a segédfüggvény felelős azért, hogy az oszlopok számmal jelölt nevét az Excelben használatos formátummá alakíthassuk át (erre a táblázatírásnál nincs szükség, azonban a beépített függvények alkalmazásánál igen). Az átalakítás jelenleg

több mint 700 oszlopig működik, ennyi bőven elég a legbonyolultabb elszámolás elkészítéséhez is.

- tablazat2csv(): a függvény CSV-formátumú, azaz pontosvesszővel elválasztott értékeket tartalmazó állományok készítését végzi. A CSV az egyik legelterjedtebb formátum, köszönhetően annak, hogy egyszerű, kisméretű, és minden táblázatkezelő támogatja. A függvény feladata a kapott adathalmaz átalakítása, ezt egyszerűen úgy végzi, hogy a különböző cellába kerülő adatokat pontosvesszővel választja el egymástól, az új sorokat pedig újsor jellel (\n) képzi.
- tablazat2xls(): a függvény az Excel alapértelmezett formátumát állítja elő. Az XLS előnye a CSV-hez képest abban rejlik, hogy lehetőséget biztosít a táblázat tetszőleges formázására (cellák egyesítése, színezés, képek használata, szövegformázás stb.), így bonyolultabb és szebb táblázatok is készíthetőek. Az XLS egy bináris állomány, azonban generálása az Interneten található, szabadon hozzáférhető segédfüggvények segítségével egyszerűen és gyorsan elvégezhető PHP-ben.

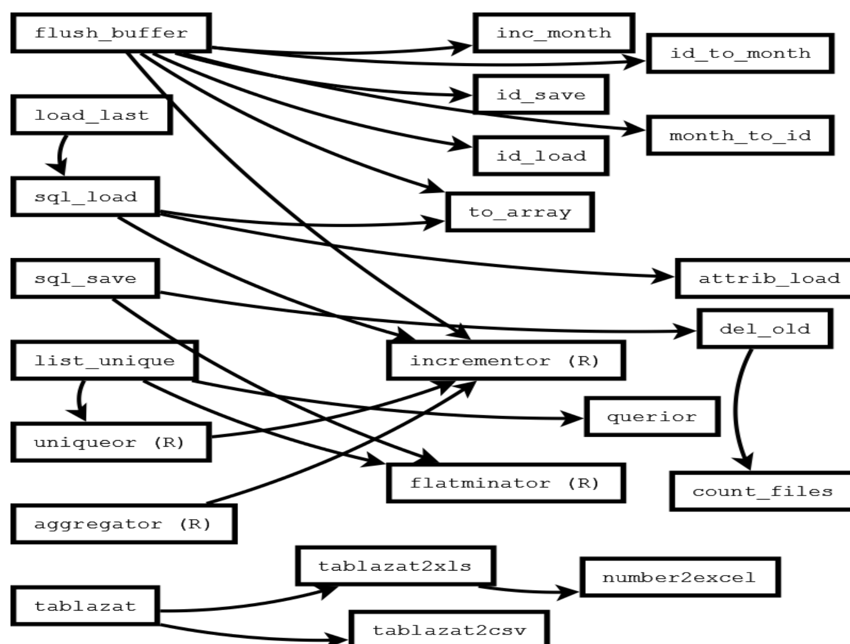
5.2.7. Egyéb függvények

A rendszer működése során fontos, hogy az esetlegesen előforduló hibákat eltároljuk, és a program futását a hibától függően akár meg is szakítsuk. Mivel az elszámolórendszer parancssorban fog futni, a felhasználó a hibákról csak egy logfile-ból fog tudni tájékozódni. Fontos továbbá megkülönböztetni a végzetes és az elhárítható hibákat, és minél pontosabban beazonosítani a hibát okozó folyamatot.

- error(): a függvény feladata a különböző függvény- vagy rendszerszintű hibák eltárolása a konfigurációs állományban beállított helyre, illetve a futás esetleges felfüggesztése. Az error() függvényt hiba esetén hívja meg minden esetben a hibát észlelő függvény vagy rendszerfolyamat. Bemeneti paraméterei a hívó

függvény neve, a hiba leírása és a hiba szintje. A rendszer jelenleg kétszintű hibakezelést használ. A függvények és segédfüggvények hibája a 0. szint, ilyenkor a program futása folytatódik. A rendszerszintű folyamatok hibája alkotja az 1. szintet, ilyenkor a futás leáll, mivel a rendszer a végzetes hibából másképp nem képes felállni. Több helyen előfordul az is, hogy hibakeresési szempontból a függvény 0. szintű hibáját rögtön követi egy 1. szintű hiba, ennek oka az, hogy így a nem várt esemény könnyebben vizsgálható, és az esetleges hiba gyorsabban elhárítható, mintha csak egy 1. szintű hiba után tesztelnénk a rendszert. A függvény neve kismértékben félrevezető, mivel nem csak hibaüzenetek kerülnek bele, hanem figyelmeztetések, illetve egyéb események is (például hónapváltás jelzése).

5.3. Összefüggések a rendszer függvényei között



5.2. ábra: A rendszer függvényeinek kapcsolata

A rendszer legfontosabb függvényei közötti kapcsolatot mutatja be az 5.2. ábra. A nyilak jelzik, hogy melyik függvény (nyíl eleje) melyik függvényt (nyíl vége)

hívja meg. A függvény nevét követő (R) jelzi, hogy az adott függvény rekurzív. (Az előző részben bemutattam, hogy a rendszer működéséhez milyen függvények szükségesek. Ahogy már korábban is írtam, az egyes függvények nem önmagukban állnak, hanem legtöbbször egymást hívják meg különböző paraméterekkel, esetenként rekurzív működésűek.)

Az ábra felépítése is mutatja, hogy a rendszer működése során rendkívül fontos a flatminator() és az incrementor() függvény, amelyeket mind a tranzakciókezelés, mind az elszámolás készítésekor több függvény használ. Rekurzív működésükből adódóan egyszerűbb és általánosabb működésre képesek, mint a nem rekurzív megfelelőjük (ami esetenként sokkal bonyolultabban lenne megvalósítható, ha egyáltalán lehetséges egy általános megoldás az adott problémára). E két függvény forráskódját tartalmazza a Függelék.

5.4. Tranzakciókezelő modul

A rendszer tranzakciókezelő modulja felelős a tranzakciók begyűjtéséért, az egyes tranzakciók megfelelő elkönyveléséért, illetve a biztonságos működés érdekében memóriamentések létrehozásáért. Továbbá lehetőséget teremt a rendszer hiba utáni újraindulására, a hibák és figyelmeztetések eltárolásával pedig segít a problémák megoldásában és elkerülésében.

A rendszer induláskor betölti a megfelelő paramétereket a konfigurációs állományból, majd megpróbálja felépíteni a többszintű fát memóriamentés alapján. Ha nincs egyetlen memóriamentés sem, vagy ha értelmezhetetlen az utolsó elkönyvelt tranzakció száma, akkor teljesen elölről kell kezdeni az indulást. Ekkor van szükség a konfigurációs állományban meghatározott account_from változóra, ami megadja, hogy ilyen esetben mikortól kezdve kell az elszámolást létrehozni.

Akár teljes újraindulás történik, akár sikerült a memóriába betölteni érvényes memóriamentés alapján a többszintű fát, a feladat minden esetben a tranzakciók fogadása és könyvelése. Teljes újraindulásnál először szükség van a month_to_id() függvényre is, hogy a rendszer megkapja az utolsó tranzakciót. A

flush_buffer() a buffer()-rel kommunikálva kezeli a tranzakciókat. A flush_buffer() függvény működését befolyásolja a konfigurációs állomány flush_interval paramétere, ami megadja, hogy milyen időközönként kell a távoli kiszolgálóhoz fordulva új tranzakciókat lekérni. Ennek megfelelő beállítása megakadályozza azt, hogy a távoli kiszolgálót folyamatosan kérésekkel bombázza az elszámolórendszer, akár a CDR működését is hátrányosan befolyásoló módon.

A rendszer biztonságát garantálja az időszakos memóriamentés, amit a save_interval változó tárol. A jól megválasztott mentési gyakorisággal megfelelő biztonság mellett kis overheadet okozunk a rendszernek. Közvetve ide kapcsolódik a files_to_keep paraméter, ami meghatározza, hogy egyszerre hány memóriamentés eltárolása lehetséges. Indokolatlanul nagy számú memóriamentés eltárolása felesleges erőforrások (merevlemez-kapacitás, overhead) pazarlásához vezethet.

A rendszer folyamatosan figyel az esetleges hónapváltásra is. Hónapváltás esetén a már új hónaphoz tartozó tranzakció nem kerül elkönyvelésre. A pillanatnyi memóriaállapot kimentésre kerül, a biztonsági memóriamentésektől független helyre, ahol az elszámoló modul a későbbiekben hozzáférhet. Ezt követően a működés tovább folytatódhat, az új tranzakciókból megkezdődhet a következő hónaphoz tartozó többszintű fa építése.

Az előbb felsorolt feladatokat a rendszer folyamatosan (illetve majdnem folyamatosan), egymás után hajtja végre. A konfigurációs állomány tartalmazza azt is, hogy milyen gyorsan kövessék egymást az egyes tranzakciókezelési hurkok (loop_sleep), illetve hogy mennyi legyen a rendszer teljes futási ideje (time_to_live). A loop_sleep megválasztásával lehetőség van az elszámolórendszer működését olyan számítógépre hangolni, ahol az elszámolórendszeren kívül más, erőforrásigényes alkalmazások is futnak (például az elszámolórendszer és a CDR egyazon hardveren vannak). A time_to_live paraméter beállításával lehetőség nyílik a rendszer futási idejét korlátozni. Ez egyrészt az előbb említett környezetben segíthet az erőforrások megfelelő kihasználtságának elérésében (meghatározott időközönként futtatjuk csak az elszámolórendszert, és akkor is csak bizonyos ideig fut). Másrészt a rendszer folyamatos és állandó futtatása

olyan esetekben sem szükséges, amikor a tranzakciók száma ezt nem teszi indokolttá (burst-ös tranzakcióérkezéseknél indokolt lehet a szakaszos működés).

Az elszámolórendszer tranzakciókezelő része tehát tulajdonképpen egy beállítható időközönként, beállítható maximális futási idővel rendelkező programrész, ami a tranzakciókat – szintén beállítható módon – lekéri, és erőforrásoktól függően megadható biztonsági szinten működtethető. A fent részletezett funkciókat egymás után, hurkokban végzi a rendszer, erre utal a tesztrendszer jelenlegi tranzakciókezelő részének neve is (loop).

Az állomány alapú és adatbázis alapú elszámolórendszerek bemutatásánál megállapítottam, hogy az egyik legfőbb hiba mindkét esetben az, hogy az adatstruktúra nem támogatja megfelelően a tranzakciók gyors könyvelését, tárolását. A tranzakciókezelő modullal kapcsolatban fontos kijelenteni, hogy a fejlesztés során elsődleges szempont volt a megfelelő struktúra használata. Az, hogy ez mennyire sikerült jól, a rendszer végső tesztelése során derül ki.

5.5. Elszámoló modul

Az elszámoló modul működése teljesen független a rendszer tranzakciókezelési feladataitól. Ami a két alrendszert összeköti az az, hogy mindkettő ugyanazokat a függvényeket használja, illetve hogy a tranzakciókezelő modul kimenete tulajdonképpen az elszámoló modul bemenete (bár megfelelő beállításokkal egy rendszertől teljesen független állomány is feldolgozható).

Az elszámolás felhasználói interakciót igénylő művelet. Ennek oka az, hogy az elszámolórendszerek működése során általában elvárás az, hogy az elszámolás így történjen. A továbbfejlesztési lehetőségek között megvizsgálom azt, hogy milyen módszerekkel automatizálható az elszámolás teljes folyamata.

Az elszámolás készítése minden esetben úgy kezdődik, hogy az adott hónap tranzakcióit tartalmazó memóriamentést betöltjük a memóriába. Ezt követően akár egyes partnereknek, akár minden partnernek elkészíthetőek az elszámolások. Ez utóbbi eset tekinthető az egyes partnerek elszámolásának egymás utáni

elvégzéseként, így tulajdonképpen teljesen ugyanúgy működhet a modul mindkét esetben.

Ha az adott hónaphoz tartozó többszintű fa már fel van építve a memóriában, elkezdődhet az elszámolás készítésének folyamata. Először szükség van egy lekérdezés összeállítására, amit a megfelelő paraméterekkel meghívott querior() függvény végez. Az elkészített lekérdezést az aggregator() függvény használja fel, a kapott lekérdezéstől függően járja be, bontja ki vagy összegzi a többszintű fa egyes ágait. Az ezután előállt, az eredetinél kevesebb szintet és elágazást tartalmazó új fát a flatminator() függvénnyel járjuk be, és készítünk egy listát minden egyes leveléhez vezető útról. Ez a lista aztán sorról-sorra feldolgozható, és az elszámolást tartalmazó állomány könnyen felépíthető.

Előfordulhat olyan eset, amikor az elszámoláshoz különböző módon aggregált fákra van szükség. A rendszer ezt is támogatja: egyrészt az eredeti, többszintű fa az aggregator() és a flatminator() függvények hatására nem változik, hiszen új változókba kerülnek az eredmények (új fa, illetve levelekhez vezető utak listája). Másrészt pedig a logikai táblázat építésénél lehetőség van több különböző tömb feldolgozására is, amik egymást semmilyen módon nem befolyásolják, és kényelmesen CSV vagy XLS állományokba írhatóak.

Az elszámolás során rendkívül fontos szerepe van a távoli kiszolgálóval való kommunikációnak. Az állomány alapú és az adatbázis alapú tesztrendszer során említettem, hogy milyen problémákat okozhat az elszámoláshoz szükséges egyéb adatok (amiket a többszintű fában nem tároltunk el) lekérése adatbázisból (túl sok tábla illesztése lassú, az egyidejű lekérdezések akár meg is béníthatják az adatbázis-kezelőt). Az új rendszer fejlesztése során figyelmet fordítottam a problémák elkerülésére, amit az alábbi módszerek segítségével végeztem.

- Joinok minimalizálása: a távoli kiszolgáló adatbázisához lehetőség szerint olyan lekérdezésekkel fordul a rendszer, amiben nincs táblaillesztés (inner join, left join, right join, outer join). Így az adatbázis-kezelő rendszer terhelése csökken, nem befolyásolja hátrányosan a CDR kezelését az új tranzakciók fogadása.

- Lockok elkerülése: a lekérdezések futtatása nem az elszámolás soronként történő feldolgozása során, hanem attól teljesen elkülönítve történik. Így az adatbázis-kezelő rendszer nem kerül olyan helyzetbe, hogy a rengeteg lekérdezés miatt előforduló lockok lelassítsák, rossz esetben megbénítsák az egész működését.
- Egyéb adatok a memóriában: a lekérdezések eredményeit (azaz az elszámoláshoz szükséges egyéb adatokat) egy vagy több változóba helyezem, és azokat a memóriában tartom végig az elszámolás készítése során. Így az elszámolás soronkénti készítésénél elég a megfelelő tömböt megfelelően megcímezni az adott tartalomhoz tartozó egyéb adatok megszerzéséhez.

A távoli kiszolgálótól a tesztrendszerben az alábbi adatok lekérése történik meg:

- a tartalom ára, részletes leírása és egyéb metaadatok;
- a forgalmazás résztvevőinek (operátor, partner, tulajdonos stb.) részesedése a bevételből;
- az elszámolás során használt valuta és az elszámolás nyelve.

Az elszámolórendszer bevételmegosztásos működése abból adódik, hogy lehetőség nyílik az elszámolások során az adott szereplőhöz tartozó bevételarány alkalmazására az elszámolandó összeg meghatározásánál. Ez az arányszám minden eladott tartalom esetén egyértelműen meghatározható, és a távoli kiszolgálótól lekérdezhető. Általában jellemző a jelenleg piacon lévő rendszerekre (főleg a mobiltartalmak szektorára), hogy a termékek egy-egy bevételmegosztás szempontjából azonos csoportba kerülnek, és minden egyes csoporthoz rendelnek annyi RS-értéket, amennyi szereplő számára az adott tartalom után járó bevételből részesedés jut. (Természetes megkötés az, hogy egy tartalomhoz tartozó RS-ek összege minden esetben száz százalékot adjon.) Az elszámolórendszerem támogatja RS-ek lekérését a távoli kiszolgálótól, és értelemszerűen használja azokat a bevételmegosztásos elszámolások készítésére.

Az elszámolás egészét tekintve kevésbé fontos részek is megvalósításra kerültek a rendszer fejlesztése során. Ilyen például a különböző valuták támogatása, az elszámolást tartalmazó állományok (különösen az XLS-formátum esetén) formázásának lehetősége, vagy az elszámolás nyelvének választási lehetősége.

5.6. Biztonsági mentés, havi mentés, betöltés

Az elszámolórendszer tranzakciókezelési részénél fontos a biztonságos működés. Ehhez nyújt segítséget a konfigurációs paraméterek közül a save_interval változó, ami megadja, milyen gyakorisággal kell szöveges állományba menteni a memóriában lévő többszintű fát, illetve magát az adatszerkezetet. A save_interval megválasztásával a rendszer biztonságos és gyors működése között megfelelő egyensúly állítható be.

A tranzakciók fogadása során a rendszer a tranzakció időbélyege alapján észreveszi a hónapváltozást. Ekkor az addig elkönyvelt tranzakciók mindegyikét tartalmazza a memóriában levő fa. Az ezen fából készített memóriamentés a havi mentés, amit az elszámolás készítésekor használ a rendszer.

A memóriamentés készítése a tranzakciók könyveléséhez képest nagyságrendekkel nagyobb időigényű feladat, mivel merevlemez-műveletet igényel. Ugyanígy a betöltés is egy időigényes funkció, azonban ez kisebb probléma a mentéshez képest, mivel betöltésre csak elszámolás készítésekor, illetve a rendszer hiba utáni újraindulásakor van szükség, kimentés pedig ennél sokkal többször történik.

5.7. Külső kapcsolódási pontok

Az elszámolórendszer szerves része a tranzakciókezelő és elszámoló modult a termék alapú tartalomkezelő rendszerhez illesztő interfész. Ennek megvalósítása a rendszerfejlesztés egyik komponense, amit az alábbiakban mutatok be.

A korábbiakban távoli kiszolgálóként hivatkozott rendszer egy, az elszámolórendszertől független logikai egység. Fizikailag lehet ugyanazon az eszközön, amin az elszámolórendszer fut, de lehetséges a rendszerek teljes elválasztása is. Míg az elszámolórendszer elsődleges feladatai a tranzakciók kezelése és az elszámolások elkészítése, addig a távoli kiszolgálón futó rendszer feladata a tranzakciók tényleges fogadása és a tranzakció tényének végleges megőrzése. Az elszámolórendszer tervezése és fejlesztése e távoli rendszer működésére nem terjed ki. A távoli rendszernek az elszámolórendszer felé néhány alapvető kívánalomnak kell megfelelnie.

- A tranzakciókat őrizze meg a rendszer az elszámolástól függetlenül. Ennek elsősorban szabályzati okai vannak (ld. 3.2. rész, chargelog bemutatása), másrészt pedig az elszámolórendszert látja el korábbi adatokkal (esetleges hiba esetén).
- Támogassa valamilyen formában a tranzakciók tömbösített átadását, az alkalmazott technológiától függetlenül biztosítson erre lehetőséget. Erre már az új tranzakciók fogadásánál is szükség van, azonban hiba esetén a rendszer újrafelállásához feltétlenül szükséges ez a funkció, különben a futási idő nagyon megnőhet. (Ha egyenként fogadjuk a tranzakciókat, az overhead miatt rossz esetben az is előfordulhat, hogy hiba után a rendszer futása nem is lesz többet valós idejű. Például soha nem ér el a rendszer a pillanatnyilag beérkező tranzakciók azonnali fogadásának állapotáig, mindig csak a már régebben beérkezetteket fogadja.)

Az elszámolórendszer fejlesztése során a távoli kiszolgáló csak névlegesen volt távoli, fizikailag ugyanazon a számítógépen futott. A tranzakciókat egy MySQL-adatbázisban tároltam. Az elszámolórendszerrel a kapcsolatot GET-

paraméterekkel és fopen() függvényhívással tartotta tranzakciófogadó egység. Ez a megoldás – a következő fejezetben részletezett biztonsági követelményekkel kiegészítve – az elszámolórendszer későbbi működése során is kívánatos és ajánlott. A rendszer továbbfejlesztési lehetőségei között fogom vizsgálni azokat a megoldási lehetőségeket, amikor a CDR és az elszámolórendszer közötti kommunikáció ettől eltérő módon működhet.

5.7.1. Tranzakciók átadása az elszámolórendszernek

A termék alapú tartalomkezelő rendszer minden egyes tartalom eladását egy-egy tranzakcióként értelmezi, amely tranzakciókat a CDR-ben tárolja el. Ez a CDR a korábban említett chargelognak felel meg, részletes bemutatására az elszámolórendszer szempontjából nincs szükség, vázlatos felépítését az 5.4. táblázat tartalmazza.

CDR része	Néhány mező felsorolása
Általános adatok	Tranzakció száma, kiszolgáló száma
Vásárló adatai	Telefonszám, telefon típusa
Vásárlás adatai	Ár, forgalmi adó
Elszámolási adatok	Brand, owner, operator, RS értékek
Tartalom adatai	Típus, leírás, állomány neve

5.4. táblázat: A CDR felépítése

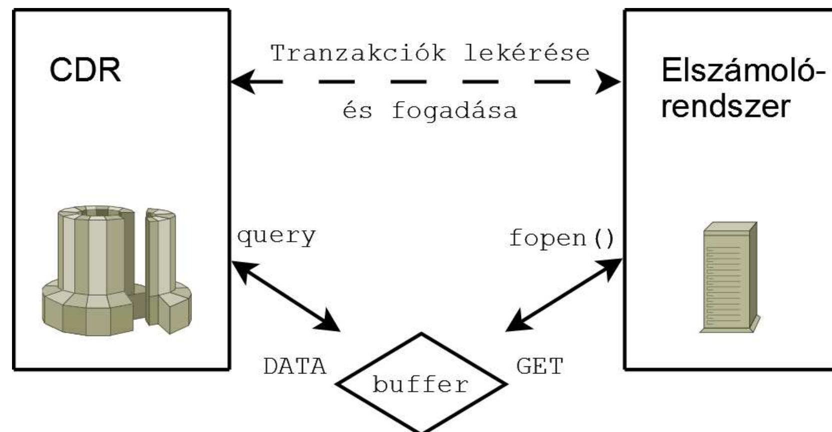
Az elszámolórendszer kiszolgálásához fejlesztem ki a buffer() függvényt, aminek feladata a távoli rendszer és az elszámolórendszer közötti kapcsolat biztosítása. Fizikailag a buffer() függvény a távoli kiszolgálón fut, és egyidejű kommunikációt folytat a CDR-rel és az elszámolórendszer tranzakciókezelő moduljával.

A buffer() akkor lép működésbe, ha az elszámolórendszer meghívja. Az általam kifejlesztett megoldásnál az elszámolórendszer GET-paraméterekkel vezérli a buffer()-t, amikor egy fopen() függvényhívással megpróbálja a buffer() később előállításra kerülő kimenetét megnyitni. Ennek előnye, hogy lehetőség van az elszámolórendszer azonosítására, így növelhető a biztonság. Az elszámolórendszer jelenleg háromféle feladatra használja a buffer()-t (tranzakciók

lekérése, év-hónap meghatározása tranzakció alapján, tranzakció meghatározása év-hónap alapján), ezek mind külön GET-paraméterben utaznak. Ezenkívül a legtöbbet használt tranzakciókérés esetén az elszámolórendszer az adatstruktúrát is a buffer() rendelkezésére bocsátja, ami azt a saját konfigurációs állományának segítségével a CDR mezőire fordítja le.

A megfelelő vezérlés után a buffer() a CDR-ből lekéri a szükséges adatokat, és az éppen ellátott funkciótól függően átalakítja azokat. Tranzakciók lekérésénél a CDR megfelelő mezőiből összeállított adatsorok egy tömbbe kerülnek, amit a beépített serialize() függvény segítségével adatfolyammá alakít a buffer(). Ezt az adatfolyamot aztán a szintén beépített unserialize() függvénnyel alakít vissza a korábban már bemutatott flush_buffer() függvény. Akár tranzakciólekérés, akár hónap- vagy tranzakciómeghatározás történt, a buffer() minden esetben az alapértelmezett kimenetre írja a visszaadni kívánt adatokat, amit aztán az őt hívó függvény az fopen() függvény segítségével olvas be. (Az fopen() beépített függvény működése lehetővé teszi, hogy ne csak helyi állományok beolvasására használjuk, hanem távoli virtuális állományokat – amilyen egy URL – is megnyithassunk olvasásra, és így távoli kiszolgálók kimeneti adataihoz juthassunk.) Ezzel a buffer() függvény futása befejeződik, a CDR és az elszámolórendszer közötti virtuális kommunikáció véget ér.

Ahogy korábban említettem, a buffer() függvény fizikailag a távoli rendszer része, azonban logikailag és működés szempontjából a lenti modell indokolt: a buffer() a két egyenrangú rendszer közötti kommunikációs csatornát valósítja meg. Az ábrán folytonos nyilakkal jelölöm a tényleges kommunikációs csatornákat, és szaggatott nyíl mutatja a kommunikáció virtuális útját, hiszen tulajdonképpen az elszámolórendszer és a CDR között történik adatmozgás. A buffer() függvény vázlatos működését mutatja a következő oldalon található 5.3. ábra.



5.3. ábra: A buffer() függvény működése

5.7.2. Tranzakciók generálása

Ugyan a végleges rendszernek nem része, azonban mindenképpen érdemes megemlíteni a tesztrendszer kapcsán a próbatranzakciók generálására létrehozott buffer_fill_cdr() függvényt. A GET-paraméterrel vezérelhető függvény a bemeneti adatban megadott számú tranzakciót generál, és azokat a CDR-be tölti. A tranzakciókat generáló kód elkészítése során figyeltem arra, hogy a próbatranzakciók életszerűek legyenek, és valóságszerűen különböző számban álljanak az elszámolórendszer rendelkezésére.

A tesztrendszer CDR-e 36 oszlopot tartalmazott, ez nagyjából megfelelő részletességgel tartalmaz minden olyan adatot, amire egy valódi, mobiltelefonos tartalmakat szolgáltató tartalomkezelő rendszer működéséhez szükség lehet. Mivel a CDR oszlopainak töredéke szükséges az elszámolás elkészítéséhez, ezért a 3. fejezetben bemutatott tesztrendszerekkel összevethető lesz az új elszámolórendszer minden futási paramétere. (A CDR 36 oszlopa és az elszámolórendszer ennél jóval kevesebb – a tesztrendszer esetén 8 – attribútuma közötti megfeleltetést a buffer() függvény látja el. A buffer() függvényt pedig a korábban bemutatott módon a flush_buffer() függvény vezérli, és kezeli a buffer() által visszaadott, az elszámolórendszer által már elkönyvelhető adatokat tartalmazó sorokat.)

5.8. Biztonsági kérdések

Minden informatikai rendszer esetén kulcskérdés a biztonságos működés. Lehet egy rendszer bármilyen jó, ha működése nem biztonságos, akkor a rendszer rossz, használata több bajt okozhat, mint amennyi hasznot hajt. Biztonság alatt egyrészt a belső biztonságot (mennyire hibatűrő a rendszer, hogyan lehet az esetleges hibákat elkerülni, bekövetkezésük esetén megoldani a kialakult helyzetet stb.), másrészt a külső biztonságot (szándékos vagy véletlen külső támadások, betörési kísérletek stb.) értjük. A belső biztonsággal részletesen az 5.9. részben foglalkozom, itt elsősorban a külső támadások lehetőségét vizsgálom.

Egy rendszer legsebezhetőbb pontja általában az, ahol a külvilággal kommunikál. Könnyen elképzelhető, hogy milyen problémákhoz vezethet, ha jogosulatlan személy által készített kód lép a másik kommunikációs partner szerepébe. Kisebb probléma – bár ez is fontos – az, ha a rendszer futása a rossz, értelmezhetetlen bemeneti adatok miatt leáll. Nagyobb gondot jelent az, ha a támadó szándéka a rendszer feltörése, jogosulatlan adathozzáférés, adatmódosítás, adattörlés. Ilyen esetekben akár helyrehozhatatlan hiba is előfordulhat, mert a program nem megfelelő elkészítése esetén a támadó olyan jogosultságokhoz juthat, amelyek segítségével gyakorlatilag bármit megtehet, amit csak akar. Egy elszámolórendszer esetén egy ilyen eset katasztrofális következményekkel járhat: nem csak az elszámolórendszert üzemeltető résztvevőnek okoz károkat, hanem az összes, az elszámolásokat használó piaci szereplőnek is (konkrét esetben például az operátoroknak és/vagy a partnereknek, vagy az egyes partnereket tömörítű ún. aggregátoroknak).

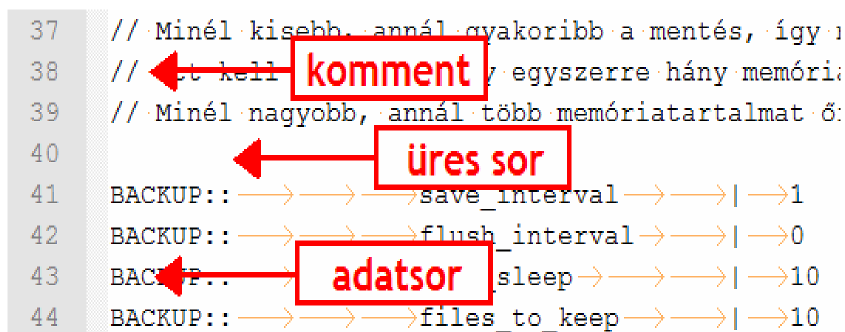
Az általam készített elszámolórendszernél két kommunikációs csatornát kell vizsgálni. A különböző állományolvasási műveleteknél kell megakadályozni, hogy a támadó által készített, kárt okozó kód ne futhasson le. Amikor a rendszer a távoli kiszolgálóval kommunikál, meg kell akadályozni, hogy bármelyik fél helyére lépve kárt okozhasson a támadó.

5.8.1. Állományolvasás

Az elszámolórendszer több esetben végez állományolvasást működése során:

- konfigurációs állomány beolvasása;
- utolsó tranzakció azonosítójának beolvasása;
- memóriamentés beolvasása.

Minden esetben hasonló védekezési lehetőség áll rendelkezésre. Az esetleges támadást úgy lehet a legegyszerűbben elkerülni, hogy felhasználjuk az olvasásra megnyitott állomány általunk ismert struktúráját. A konfigurációs állomány üres sorokból, kommentekből és adatsorokból áll. Ezek közül csak az adatsorokkal kell foglalkozni, hiszen a többit figyelmen kívül hagyjuk. Az adatsorok esetén pedig kihasználhatjuk, hogy egy sor felépülése mindig illeszkedik a következő sablonra: „<típus>:: <név> | <érték>”. Ennek tudatában a védekezés a sor felépítésének vizsgálatával, illetve a szöveges adatok beépített függvényekkel (htmlspecialchars(), addslashes() függvények) történő kezelésével megoldható. A konfigurációs állomány felépítését mutatja be az 5.4. ábra.



5.4. ábra: A konfigurációs állomány felépítése

Az utolsó tranzakció azonosítóját tartalmazó állomány esetében még egyszerűbb a feladat. Az állomány beolvasása után mindössze arra van szükség, hogy a tartalmat egy egész számmá konvertáljuk, ezzel az esetleges támadás teljes egészében kivédhető és felismerhető. (Ebben az esetben figyelni kell arra,

hogy egy ilyen konvertálás esetén könnyen keletkezhet értelmezhetetlen érték – például nulla vagy egy negatív szám.)

A memóriamentés beolvasása hasonlóan végezhető, mint a konfigurációs állomány esetén. Ebben az esetben is felhasználhatjuk azt, hogy a sorok felépítése ismert. Memóriamentésnél két típusú adatsor fordul elő: a deklarációs rész (a CREATE TABLE utasítás után) és a ténylegesen adatokat tartalmazó rész (az INSERT INTO utasítást tartalmazó sorok). Az állományt a rendszer állítja össze (szabványos SQL-formátumú állomány), így felépítése teljesen ismert, így az esetleges támadás ebben az esetben is elhárítható. Az állományban előfoduló megjegyzések szintén nem okozhatnak problémát, mivel azokat a beolvasás során figyelmen kívül hagyjuk.

5.8.2. Kommunikáció

Más rendszerekkel vagy felhasználóval való kommunikációra az alábbi esetekben kerül sor:

- tranzakcióhoz tartozó év – hónap lekérdezése;
- év – hónaphoz tartozó tranzakció lekérdezése;
- tranzakciók lekérdezése;
- elszámolás felhasználói felülete.

Az első két eset tulajdonképpen megegyezik abból a szempontból, hogy egy rövid GET-paraméteres vezérlésre egy rövid, fopen() függvénnyel fogadható kimenet érkezik válaszként. A tranzakciók lekérdezése hasonlóan történik, csak ott a vezérlés valamivel bonyolultabb (elküldjük az adatstruktúrát is), illetve a kimenet jóval nagyobb mennyiségű adatot tartalmaz. A jelenlegi megvalósításban azonban mindhárom kommunikációs megoldás esetén az esetleges támadás hasonló módszerrel előzhető meg. Egyrészt lehetőség van egyszerű azonosítás megvalósítására: GET-paraméterben, illetve az fopen() függvénnyel kezelt kimenetben is elküldhetünk egy-egy előre egyeztetett jelszót, aminek hiányában a kommunikáció megszakításra kerül (feltételezett támadás miatt). Másrészt

növelhető a biztonság HTTP helyett HTTPS alkalmazásával, amikor az átvitt adatok titkosításra kerülnek, így egy esetleges passzív támadásnál (lehallgatás) a támadó nem tud értelmezhető adatokhoz hozzáférni. Ez utóbbi védelemnél azonban számolni kell a kommunikáció sebességének csökkenésével. További védekezési lehetőségekre is van mód, azonban ezek már alacsonyabb rétegekben kerülhetnek megvalósításra. Ilyen lehetőség a távoli kiszolgáló és az elszámolórendszer közvetlen összekötése, amikor kizárható az esetleges külső támadás. Másik megoldás az IP-cím és a kommunikációs port alapján történő szűrés a rendszerek között. Mivel ezek inkább hardveres és operációs rendszer szintű feladatok, további ismertetésüktől eltekintek.

5.9. Hibakezelés

A rendszer biztonságos működése nem csak a külső támadóval szembeni védekezésen múlik. Sokkal gyakrabban fordulhatnak elő hibák a rendszeren belül, pusztán azért, mert míg például memóriamentés alapján történő indulásra csak heti-havi rendszerességgel van példa, addig a faépítés során több tíz- vagy százezer műveletet végez a rendszer – másodpercenként. A hibakezelés során a feladat nem csak az, hogy a rendszer észlelje és kezelje a futás során felmerült hibákat, hanem lehetőség szerint mindent meg kell tenni annak érdekében, hogy a hibákat teljesen elkerülhessük. Erre egyrészt futási időben van szükség: különböző monitorozási megoldásokkal a rendszer működését úgy kell irányítani, hogy a hiba bekövetkezési valószínűsége a lehető legkisebb egyen. Másrészt viszont futási időn kívül is szükséges a hibák megelőzése: már a rendszer tervezése-fejlesztése során figyelni kell arra, hogy a hibák előfordulási lehetőségét a lehető legkisebbre szorítsuk vissza.

A rendszer tervezésekor a különböző függvények kommunikációja paraméterátadásokkal és globális változókkal történik. (Globális változó alatt olyan változókat értek, amikre az egyes függvények a global utasítással hivatkoznak. Ez nem összekeverendő a PHP „register_globals ON” beállításával, amit természetesen a program fejlesztése során nem alkalmaztam. A

register_globals veszélyeiről bővebben: [17].) A globális változók mindig magukban hordozzák annak a lehetőségét, hogy két konkurens függvény ugyanazt az adatot módosítva hibás működést okoz. Éppen ezért globális változókat csak azokban az esetekben használok, amikor feltétlenül szükségesek, minden egyéb esetben paraméterekkel kommunikálnak a függvények.

Az 5.2.7. részben bemutatott error() függvény a futási idejű hibakezelésért felelős. A hibát észrevéve a függvény egyrészt minden esetben eltárolja a hibaeseményt, másrészt a program futását leállítva megszakíthatja a hibás működést. A függvény egyszerűen továbbfejleszthető úgy, hogy tetszőleges további kommunikációs csatornán (e-mail, IM stb.) tájékoztassa a felhasználót a bekövetkező hibákról. A hibaeseményeket tartalmazó állományt vizsgálva lehetőség van a hiba mielőbbi megszűntetésére, és a program újraindítására. Ez a megoldás elkerülhetővé teszi a rendszer rendelkezésére álló erőforrások felesleges kihasználását, hiszen a rendszer megvalósításától függően lehetséges, hogy a hibás működés okozta terhelés más, nem elszámoláshoz köthető funkciók ellátását befolyásolja hátrányosan.

6. Tesztelés

Az elszámolórendszert az egyes függvények és modulok fejlesztése során is folyamatos teszteknek vetettem alá, hogy az éppen fejlesztés alatt álló rész működése a kívánalmaknak megfelelő legyen. Igyekeztem minden esetben az optimális megoldást megtalálni, hogy a futási idő is lehetőleg minimalizált legyen. A függvények tesztelése során főleg szintaktikai hibákat kerestem, persze ekkor sem feledkeztem meg a más jellegű hibák felkutatásáról és kijavításáról.

A teljes rendszer tesztjei során tehát elsősorban a rendszerszintű működés vizsgálatára van szükség, az esetleg előforduló szemantikai hibák kiszűrésével. Ekkor már nem kell a szintaktikai hibákra koncentrálni, hanem a teljes működés helyességét kell ellenőrizni. Amikor az elszámolórendszer hibátlan működése már biztosított (funkcionális teszt) – nagy terhelés alatt is (terheléses teszt), akkor következhet a teljesítményteszt, amely során az új rendszert a tesztrendszerekkel, illetve egy már használatban lévő elszámolórendszerrel vetem össze.

- Funkcionális teszt: a funkcionális teszt során azt vizsgálom, hogy a rendszer a kívánalmaknak megfelelően működik-e.
- Terheléses teszt: a terheléses teszt során a rendszer megbízhatóságát vizsgálom nagy terhelés hatása alatt.

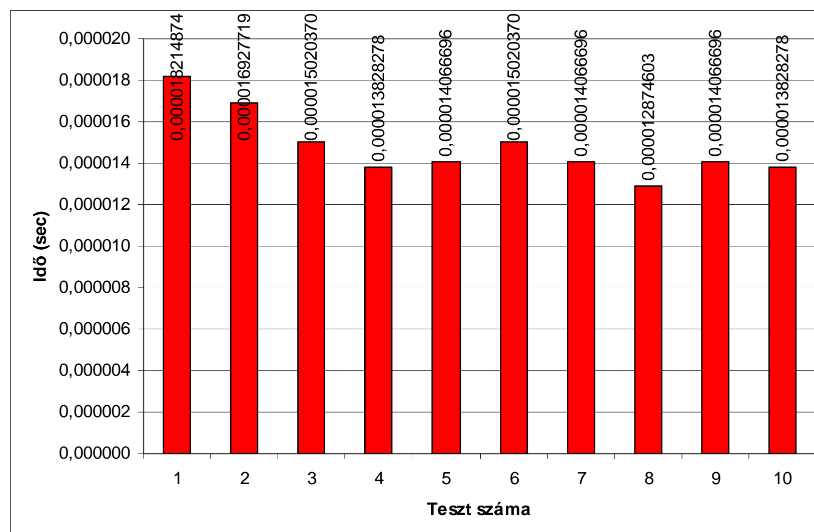
A rendszer a funkcionális teszt alatt hibátlanul működött, a megvalósított feladatokat minden esetben úgy látja el, ahogy az a rendszertől elvárható.

A terheléses teszt során az olyan jellegű hibák megkeresése és kijavítása történik meg, amik kis terhelés mellett jellegükből adódóan nem derülnek ki. Egy informatikai rendszer (is) másképpen viselkedik csúcsterhelés mellett, mint ahogy azt az átlagos terhelés esetén elvárnánk. A rendszer a terheléses teszt során is mindvégig megbízhatóan működött, nagyszámú tranzakció fogadása alatt sem omlott össze, köszönhetően az előrelátó tervezésnek.

6.1. Összehasonlítás a tesztrendszerekkel

A 3. fejezetben bemutatott tesztrendszereken háromféle tesztet végeztem. Az első tesztsorozatban a rendszerek tíz véletlenszerűen kiválasztott tartalom aktuális letöltési számát számolta ki. A második tesztsorozatban tíz új tranzakció elkönyveléséhez szükséges időket vizsgáltam (az adatbázis alapú rendszernél különválasztva a már létező és a még nem létező tartalmakat). A harmadik tesztsorozat folyamán pedig egy egyszerűsített elszámolást végeztem el a rendszerekkel, az ezekhez tartozó futási időket figyelve. (Az elszámoláshoz szükséges egyéb adatok lekérése e tesztnek sem része, az ezzel kapcsolatos problémák megoldását az 5.5. részben mutattam be. Az eredmények összemérhetőségét az garantálja, hogy egyik rendszer tesztje során sem vettem figyelembe ezt a problémaforrást, hanem külön tárgyaltam, és új megoldási lehetőséget találtam rá.) A tesztrendszerekkel való összehasonlítás a rendszer teljesítménytesztje is egyben.

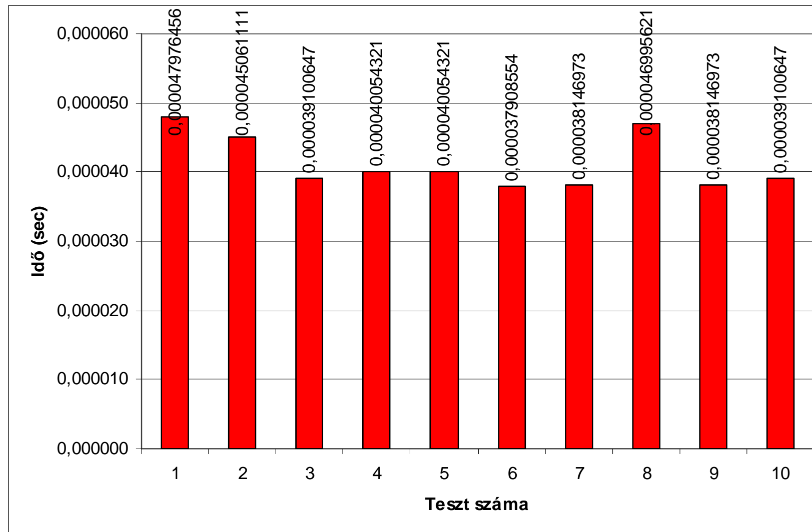
Az első teszt eredményeit a 6.1. ábra tartalmazza.



6.1. ábra: Az első teszt eredményei (memória alapú elszámolórendszer)

Az első teszt időeredményei tíz mikroszekundomos nagyságrendűek. A legkisebb érték (13 μsec) a nyolcadik, a legnagyobb (18 μsec) az első esetben fordult elő. Az átlagos idő 15 μsec, a szórás 1,6 μsec.

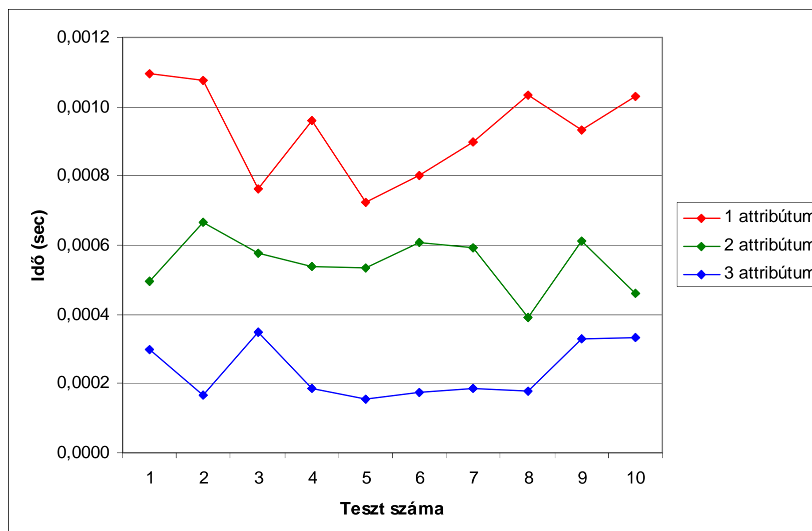
A második teszt eredményeit a 6.2. ábra tartalmazza.



6.2. ábra: A második teszt eredményei (memória alapú elszámolórendszer)

A második teszt időeredményei tíz mikroszekundomos nagyságrendűek. A legkisebb érték (38 μsec) a nyolcadik, a legnagyobb (48 μsec) az első esetben fordult elő. Az átlagos idő 41 μsec, a szórás 4 μsec.

A harmadik teszt eredményeit a 6.3. ábra tartalmazza.



6.3. ábra: A harmadik teszt eredményei (memória alapú elszámolórendszer)

A tesztesetek futási ideje – négy esettől eltekintve – legfeljebb száz mikroszekundumos nagyságrendű. A legkisebb futási időt (155 μsec) a három

attribútumos ötödik esetnél, a legnagyobbat (1095 μsec) az egy attribútumos első esetnél mértem, az átlagos idő 571 μsec, 306 μsec szórással.

Az elszámolórendszert végül összevetettem a tesztrendszerekkel a mért futási idők alapján. Az eredményeket a 6.1. táblázat tartalmazza.

Teszt	Áll. / Mem.	A.b. / Mem.
1. teszt	0,0113 %	1,9764 %
2. teszt	83,6318 %	1,5044 %
3. teszt, 1 attribútum	0,6900 %	123,9439 %
3. teszt, 2 attribútum	0,3745 %	24,8098 %
3. teszt, 3 attribútum	0,1608 %	10,5389 %

6.1. táblázat: Az egyes teszteredmények összehasonlítása

Ahogy az a táblázat értékeiből is kiolvasható, az új elszámolórendszer egy esettől eltekintve minden tesztnél jobban szerepelt, mint a tesztrendszerek. Az állomány alapú tesztrendszer az ötből négy alkalommal legalább négy nagyságrenddel nagyobb futási időt produkált, mint az új elszámolórendszer. Az első két tesztnél két nagyságrenddel volt lassabb az adatbázis alapú elszámolórendszer, mint az új elszámolórendszer.

Az új rendszer előnyei leginkább az elszámolás esetén fontosak, hiszen ez egy igen időigényes feladat. Látható, hogy az attribútumok számának növelésével a memória alapú elszámolórendszer mind nagyobb különbséget produkál a másik két rendszerhez képest. Valós körülmények között nem ritka a négy, öt vagy ennél is több attribútum alapján történő elszámolás készítése, így az új rendszer a teszteredmények alapján biztosan legalább egy nagyságrendes gyorsulást eredményezhet.

6.2. Összehasonlítás egy működő rendszerrel

Az általam fejlesztett elszámolórendszer célja, hogy a jelenleg működő, állomány alapú vagy adatbázis alapú elszámolórendszerekkel szemben egy teljesítményét és megbízhatóságát tekintve jobb alternatíva legyen. A rendszerem sok beállítási és finomhangolási lehetőségével szinte bármelyik termék alapú

tartalomkezelő rendszerhez illeszthető. Az előzőekben ismertetett teszteredmények alapján kijelenthető, hogy a memória alapú elszámolórendszer legalább egy nagyságrenddel csökkentheti az elszámoláshoz szükséges időt. Hogy ez konkrétan mit jelenthet, azt egy éles tartalomkezelő rendszer és egy abba integrált elszámolórendszer segítségével mutatom be.

Adott egy mobiltelefonos tartalmakat kínáló cég. Funkcióját tekintve elsősorban e tartalmakat értékesítő cégek (partnerek) számára nyújt szolgáltatásokat, szerepe értelmezhető összekötő kapocsként a mobiltelefonos szolgáltatók (operátorok) felé. További szereplőként jelen vannak több partnert összefogó cégek (aggregátorok), valamint jogvédő irodák is. A közvetítő cég elszámolórendszere egy állomány alapú rendszer, amely támogatja a bevételmegosztást is. Mivel a tartalmak forgalma a mobiltelefonok elterjedésével az utóbbi években robbanásszerűen megnőtt, a korábban jól és viszonylag gyorsan működő elszámolás időtartama jelenleg órákban mérhető. A lassúság oka a korábbiakban ismertetett problémákra (nem optimális adatszerkezet, sokszoros joinok, nagyszámú lock) vezethető vissza.

Az új, memória alapú elszámolórendszer egyszerűen kiválthatja a jelenleg működő elszámolórendszert. Főleg egyszeri beállításokat kell elvégezni a tartalomkezelő rendszeren, hogy az átállás lehetséges legyen. A jelenlegi chargelogos tranzakciókezelés egyszerűen kiegészíthető egy CDR-rel, ami a tartalomkezelő rendszer és az elszámolórendszer közötti kapcsolatot biztosíthatja. A CDR töltése egyetlen függvényhívással megoldható. A távoli kiszolgálón elvégzett beállítások után az adatátvitelért felelős buffer() függvény paraméterezése is gyorsan megvalósítható, majd az elszámolórendszer beállítása után gyakorlatilag kész is az integráció. Mivel az elszámolórendszerem maga nem kifejezetten nagy erőforrásigényű program, biztonsági okokból is jó megoldás a régi és az új elszámolórendszer egyszerre való alkalmazása – legalábbis a kezdeti időkben. Ezt követően pedig szintén minimális beállítás után teljesen átveheti a régi elszámolórendszer összes feladatát az új.

Az új elszámolórendszerre való átállás – a körülmények alapos feltérképezése és az igények felmérése után – minden termék alapú

tartalomkezelő rendszer elszámolórendszerénél ajánlott. A rendszerem kész minden elszámolással kapcsolatos feladat elvégzésére – az eddigieknél jóval gyorsabban.

7. Továbbfejlesztési lehetőségek

Ebben a részben röviden bemutatom, hogy az általam fejlesztett új, puha valós idejű elszámolórendszer fejlesztése során milyen kompromisszumokat kellett hoznom, ezeket hogyan lehetne a rendszer egy későbbi verziójában kijavítani, illetve milyen további lehetőségeket látok a rendszer fejlesztésében.

7.1. Kompromisszumok a fejlesztés során

Egyetlen informatikai rendszer sem tökéletes. Egyrészt soha sem készülne el, hiszen mindig vannak hibák, amiket ki kell javítani; mindig vannak algoritmusok, amiket tovább lehet fejleszteni; és mindig vannak funkciók, amikkel még ki kellene egészíteni a rendszert. Ebből adódik az, hogy a rendszer gyakorlatilag korlátlan idő- és energiabefektetést igényelne. Harmadrészt pedig minél több feladatot lát el a rendszer, minél gyorsabban és tökéletesebben akarunk minden szükséges funkciót ellátni, annál nagyobb erőforrásigényű lesz a rendszerünk. A fejlesztés lényege a fenti szempontok (idő, energia, funkciók, gyorsaság, tökéletesség, erőforrásigény) közötti kompromisszumok meghozása, és a körülményekhez képest a lehető legjobb rendszer létrehozása.

Az általam készített új, puha valós idejű elszámolórendszer is egy informatikai rendszer, így nem lehet, és nem is lett tökéletes. A fejlesztés során az alábbi problémákat nem sikerült maradéktalanul megoldani, illetve ezen esetekben lehetne a jelenleginél jobb megoldásokat keresni.

- Algoritmusok: a jelenleg használatos algoritmusok elkészítése során törekedtem a minél gyorsabb futási időre, azonban biztos létezik néhány olyan megoldás, amik a rendszer jelenlegi sebességét tovább növelhetik. Főleg a rekurzív függvények esetén érdemes a

függvényeket optimalizálni, mivel a sok függvényhívás miatt azokban az esetekben számottevő javulást lehetne elérni.

- HTTPS kommunikáció: a rendszer jelenleg a távoli kiszolgálóval HTTP-n keresztül kommunikál, a HTTPS-es megoldás még nem készült el. Mivel az azonosítás már így is megoldott, és a kommunikációs csatorna lehallgatásának esélye nulla (az elszámolórendszer ugyanazon a gépen fut, mint ahol a CDR található), ez nem jelent komoly problémát. A HTTPS-en keresztüli kommunikáció egyébként viszonylag egyszerűen megvalósítható, és kellően nagy adatátviteli sebesség mellett nem okoz számottevő lassulást.
- Több konfigurációs paraméter: a rendszer futását jelenleg nagyjából 20 paraméter befolyásolja. Ezek segítségével a rendszer testreszabhatósága meglehetősen nagymértékben garantált. Azonban néhány további beállítási lehetőségre is szükség lehet a későbbiekben. Ilyen paraméter lehetne a program számára maximálisan rendelkezésre álló memória nagysága (jelenleg a felső határ a PHP számára biztosított mennyiség), vagy az egyes függvények és segédfüggvények maximális memória- és időigénye. Esetleg érdemes lenne a tranzakciókezelő és az elszámoló modul beállításait különválasztani. Ezzel a lehetőséggel lehetőség nyílna a pillanatnyilag könyvelt és az elszámolandó tranzakciók közötti függés teljes megszüntetésének (ennek lehetőségeiről bővebben a következő részben).
- Kommunikáció a CDR-rel: a jelenlegi megvalósítás egy GET-paraméretes azonosítású és vezérlésű fopen() függvényhívás. Ezenkívül érdemes lehet egyéb kommunikációs csatornákat használni, például e-mailen keresztül történő adatcsere. Érdekes lehet továbbá egyéb kommunikációs protokollok használata (FTP, FTPS, SSH), vagy olyan, más esetekben már elterjedt megoldások támogatása, mint amilyen a VPN. Más kommunikációs módokat

akkor érdemes használni, ha azok nagyobb adatátviteli sebességet (amit ki is tudunk használni az elszámolás gyorsításához), vagy biztonságosabb adatátvitelt tesznek lehetővé a jelenlegi HTTP-s megoldáshoz képest. Főleg a biztonság lehet fontos szempont, mivel az elszámolórendszer gyorsasága jelenleg egyáltalán nem az adatátvitel sebességtől függ.

7.2. Továbbfejlesztési lehetőségek

Egy infomatikai rendszer fejlesztése általában nem ér véget a rendszer elkészültével. Érdemes néhány további lehetőséget összeszedni, amik az elszámolórendszert értékes funkciókkal egészíthetik ki. Az alábbi, inkább távlati megoldásokat lenne érdemes megvalósítani.

- Hibák, figyelmeztetések, egyéb események jelzése: mindenképpen érdemes lehet kibővíteni a rendszert e-mailen vagy IM-programokon keresztül történő kommunikációra. A nem várt eseményekről jelenleg csak a hibakezelő állományból lehet értesülni, ami esetleg problémát okozhat. E-mailes vagy IM-es értesítésnél a hiba elhárítása gyorsabban megkezdődhet.
- Automatikus elszámolás: ugyan az elszámolórendszerekkel szemben általában elvárás az elszámolás felhasználói beavatkozással történő elkészítése, lehetséges azonban a teljes automatizálás is. Ebben az esetben az elszámolásokat a rendszer teljesen önállóan készíti, esetleg el is küldi a partnereknek.
- Egyszerűsített memóriamentések: a rendszer a memóriamentéseket egy szabványos SQL állományba írja, és a betöltés is ebből történik. Ez sok előnyt jelent (ld. 4.3.3. rész), azonban viszonylag lassú, és felesleges overheadet jelenthet. (Főleg azokban az esetekben, amikor a struktúra nem változik hosszabb időn keresztül, valamint a hibátlan működésről megbizonyosodva már nem szükséges például

egy MySQL-rendszerrel ellenőrizni a memória tartalmát a mentések alapján.) Minimális továbbfejlesztéssel megoldható az, hogy a memóriában levő, többszintű fát reprezentáló tömb serialize() függvényhívás után előálló formáját írjuk egy állományba, és a betöltés hasonlóan, egy unserialize() függvényhívással történjen. Ezzel jelentősen javulhat a rendszer sebessége, bár megfelelő beállítások mellett a mostani megvalósítás sem okoz túl nagy overheadet.

- Sávos RS-támogatás: a rendszer jelenleg nem támogatja a sávos RS-eket, ám viszonylag egyszerűen felkészíthető. A sávos bevételmegosztás lényege az, hogy egy tartalomhoz vagy tartalomcsoporthoz tartozó RS-eket befolyásolja az, hogy az adott tartalomból mennyi került értékesítésre (tipikusan minél több tartalmat adnak el, annál nagyobb a partner részesedése, hiszen kisebb járulékos költségek mellett nagyobb forgalom realizálódik). Igazából a sávos RS alkalmazása nem is annyira az elszámolórendszer elszámoló modulját érintené. Szükség lenne természetesen egy tartalomcsoport-összegzésre a szükséges RS-ek meghatározásához és kiválasztásához, azonban elsősorban a távoli kiszolgáló átalakítása szükséges ehhez a funkcióhoz. Előfordulhat olyan eset is, amikor az RS-ek sávjai nem (csak) az eladott tartalmak számának függvénye, hanem egyéb paraméterek befolyásolhatják (például az összes forgalom), a rendszert erre is érdemes lenne felkészíteni.
- Felhasználói felület fejlesztése: érdekes lehet egy olyan felhasználói felület (UI) létrehozása, amin keresztül a felhasználónak lehetősége nyílik a rendszer pillanatnyi állapotának megtekintésére. Például láthatja a többszintű fa tartalmát valamilyen szempontból, esetleg lehetőség van részleges elszámolás megtekintésére is. További lehetőség a teljes elszámolás hó közben történő becslése, ami segítséget nyújthat a partnerek számára.

- Elszámoló modul és tranzakciókezelő modul szétválasztása: a rendszer két moduljának szétválasztása lehetővé tenné azt, hogy a tranzakciók kezelése és az elszámolás készítése teljesen függetlenül működjön egymástól. Így lehetőség nyílna arra, hogy ugyanazon rendszer két (vagy több) teljesen különböző tartalomkezelő rendszert is kiszolgáljon. Ugyanígy egyszerre több fa építésére és bővítésére is lehetőség lenne. Ha a két modul szétválik, lehetőség lenne egy központi programmag létrehozására is, ami tökéletesen optimalizálva tartalmazhatná azokat a függvényeket, amiket mindkét modul használ.

- Elszámolóközpont: az előző ötlet továbbgondolásával elérkezhetünk oda, hogy lehetőség nyíljon egy teljes elszámolóközpont megalkotására. Egyetlen ilyen központi rendszer akár több ezer másik rendszer kiszolgálását is elvégezhetné. Az elszámolóközpont tetszőleges termék alapú tartalomkezelő rendszer számára nyújthat komplex, részletesen paraméterezhető szolgáltatásokat. Az egyes tartalomkezelő rendszerek adminisztrátorai egy adminisztrációs felületen keresztül módosíthatnák az általuk igénybe vett szolgáltatások paramétereit. Egy központi rendszer végezhetné a különböző tartalomkezelőknek nyújtott szolgáltatások minőségbiztosítását, így lehetőség lenne meghatározott szolgáltatásminőség (QoS) garantálására. Az elszámolások történhetnének automatikusan, vagy felhasználói beavatkozás hatására. Lehetőség lenne különböző elszámolócsomagok választására, amiktől az előbbieken kívül a sebesség és elszámolási állományok minősége (például grafikonokat is tartalmazó PDF) is függhetne. Jelenleg, amikor a sávszélességnek és az erőforrásoknak a korábbiaknál lényegesen alacsonyabb költségeik vannak, egy jó minőségű és megbízható elszámolóközpont létrehozása rentábilis és profitábilis lehet.

8. Összegzés

Az Internet- és a mobiltelefon-felhasználók száma folyamatosan nő: az elmúlt egy évtizedben számuk nagyjából egy nagyságrenddel nőtt. A hatalmas kereslet hatalmas kínálatot generált, ami aztán további tömegeket vont be az Internet és a mobiltechnológia használatába. Mindkét szektorban kialakult egy olyan helyzet, amely a tartalomkezelő rendszerek használatának kedvezett: az Interneten a webes portálok, a mobil világban a mobiltelefonos tartalmakat nyújtó rendszerek vesznek igénybe mind több és több CMS-t.

Ez a környezet ideális elszámolórendszerek alkalmazására, amelyek gyakorlatilag tetszőleges, termék alapú CMS számára nyújthatnak komplex szolgáltatásokat. A mobiltelefonos termékek különleges piaca esetén például a bevételmegosztásos elszámolás támogatása is megoldható.

A jelenleg létező, állomány alapú vagy adatbázis alapú elszámolórendszerek kezdik elérni teljesítőképességük végső határait. Egy-egy elszámolás akár órákig vagy napokig is futhat, a rendszerek nem optimális felépítése, az elszámoláshoz szükséges adatok begyűjtése vagy pusztán a technológiai korlátok miatt.

Az általam elkészített dolgozatban először ismertettem a fenti környezet kialakulásához vezető utat: bemutattam az Internet és a mobiltechnológia fejlődését, valamint a jelenlegi helyzetet. Ez a kontextus járult hozzá a tartalomkezelő rendszerek minden korábbit felülmúló elterjedéséhez. Ezután megvizsgáltam a jelenleg létező elszámolórendszerek technológiai megoldásait, és mérések segítségével bemutattam a legfontosabb problémákat.

Az elszámolórendszerek hibáinak megismerése után elvégeztem egy új, puha valós idejű, termék alapú, bevételmegosztásos elszámolórendszer tervezését, amely a korábbi megoldásokkal szemben memória alapú működésen

alapult. A tervezés után elvégeztem a rendszer fejlesztését, amit szintén részletesen taglal jelen dolgozat.

Az elkészült rendszer tesztelése során megállapítottam, hogy ezen elszámolórendszer alkalmazásával legalább egy nagyságrenddel csökkenthető egy-egy elszámolás elkészítéséhez szükséges idő. Ez egy olyan eredmény, amely gazdaságossá teszi bármely régi elszámolórendszer lecserélését az új elszámolórendszerre.

A tesztelést követően megvizsgáltam, hogy a rendszer milyen további funkciókkal bővíthető, milyen továbbfejlesztések megvalósításával lehet még jobb, még gyorsabb, még komplexebb szolgáltatásokat nyújtani a dinamikusan növekvő tartalomkezelő rendszerek számára.

Az elszámolórendszerem jelenleg még nem éles környezetben működik, ám hamarosan konkrét piaci alkalmazása is megvalósulhat. Minden bizonnyal ekkor is meg fogja állni a helyét, és nem csak a teszteseteknél múlja felül az állomány alapú és az adatbázis alapú elszámolórendszereket.

VI. Függelék

```
function aggregator($felt, &$tomb, $curr) {
 global res_tomb;
 if(count($felt) != 0) {
  foreach($tomb as $kulcs => $ertek) {
   if($felt[0] == "*") {
    aggregator(array_slice($felt, 1),
               $ertek,
               $curr);
   }
   elseif($felt[0] == "{") {
    aggregator(array_slice($felt, 1),
               $ertek,
               array_merge($curr, array($kulcs)));
   }
   elseif($felt[0] == $kulcs) {
    aggregator(array_slice($felt, 1),
               $ertek,
               array_merge($curr, array($kulcs)));
   }
  }
 }
 else {
  incrementor($curr, $res_tomb, $tomb);
 }
}
```

```
function incrementor($koord, &$tomb, $value = 1) {
 if(count($koord) == 0) {
  $tomb[0] += $value;
 }
 elseif(count($koord) == 1) {
  $tomb[$koord[0]] += $value;
 }
 else {
  incrementor(array_slice($koord, 1),
              $tomb[$koord[0]],
              $value);
 }
}
```

VII. Irodalomjegyzék

[1] M. Achour, F. Betz, A. Dovgal, N. Lopes, P. Olson, G. Richter, D. Seguy, J. Vrana, fordítók: Bagi L. L., Csontos A., Heilig Sz., Hojtsy G., Kontra G., Papp Gy., Tóth A., Varanka Z. (a továbbiakban: PHPgroup): PHP kézikönyv (http://hu.php.net/manual/hu/)

[2] V. Cerf: How the Internet Came to Be (http://netvalley.com/archives/mirrors/cerf-how-inet.html)

[3] CMSWorks: The Web CMS Report (CMSWorks, Inc., ISSN 1551-5893 07)

[4] C. Coffman, R. Jesty, N. Walton, D. Winterbottom: Mobile Content and Services: Leveraging converging channels for content delivery (6th Edition, Informa UK Ltd., December 2006, ISBN 978 1843 116189)

[5] L. Downes, C. Mui: Unleashing the Killer App: Digital Strategies for Market Dominance (Harvard Business School Press, 2000, 2611)

[6] B. Doyle: How many CMS are there? (http://lists.cms-forum.org/pipermail/cms/20050817/001067.html)

[7] Gajdos S.: Adatbázisok (Műegyetemi Kiadó, 2006, 55053)

[8] GSM World, GSM Association Press Release (http://www.gsmworld.com/news/press_2005/press05_21.shtml)

[9] P. Gulutzan, T. Pelzer: SQL Teljesítményfokozás (Kiskapu, 2003, ISBN 963 9301 69 8)

[10] F. Heart, A. McKenzie, J. McQuillian, D. Walden: ARPANET Completion Report (Bolt, Beranek and Newman, Burlington, MA, January 4, 1978)

[11] ISC, Inc.: Internet Domain Survey (http://www.isc.org/index.pl?/ops/ds/)

[12] Katona Gy. Y., Recski A., Szabó Cs.: A számítástudomány alapjai (Typotex, 2002, ISBN 963 9326 24 0)

[13] J. Maynard: Modular Programming (Auerbach Publishers, 1972, ISBN 0877691290)

[14] Miniwatts Marketing Group: World Internet Users and Population Stats (http://www.internetworldstats.com/stats.htm)

[15] OpenSourceCMS project (http://www.opensourcecms.com/)

[16] J. O'Donnell: More shoppers take online plunge; sales hit $670 million in record day (USA TODAY Money (Online), December 17, 2006)

[17] PHPgroup: Globálisan is elérhető változók (Register Globals) használata (http://hu.php.net/register_globals)

[18] Wikipedia: Content management system (http://en.wikipedia.org/wiki/Content_management_system)

[19] Wikipedia: GSM Communications (http://en.wikipedia.org/wiki/Global_System_for_Mobile_Communications)

[20] Wikipedia: World Wide Web (http://en.wikipedia.org/wiki/World_Wide_Web)

[21] Wikipédia: Internet (http://hu.wikipedia.org/wiki/Internet)

[22] Wikipédia: Tartalomkezelő rendszer (http://hu.wikipedia.org/wiki/Tartalomkezel%C5%91_rendszer)

[23] 3G Americas, Q4 2006 Global Updates (http://www.3gamericas.org/English/Statistics)

VIII. Rövidítések

ARPANET – Advanced Research Projects Agency Network

Blog – WebLog (internetes napló)

CEPT-GSM – European Conference of Postal and Telecommunications Administrations, Groupe Spécial Mobile

CDR – Charge Data Record (vásárlási adatrekord)

CMS – Content Management System (tartalomkezelő rendszer)

CSV – Comma Separated Value (vesszővel elválasztott értékek)

DARPA – Defence Advanced Research Project Agency (Fejlett Védelmi Kutatási Programok Ügynöksége)

EDGE – Enhanced Data Rates for GSM Evolution (Növekvő Adatforgalom Fejlesztő Rendszer)

ETSI – European Telecommunications Standards Institute (Európai Távközlési Szabványosítási Intézet)

FTP – File Transfer Protocol (adatállomány átviteli protokoll)

GPRS – General Packet Radio Service (csomagkapcsolt rádió-adat szolgáltatás)

GSM – Global System for Mobile Communication (földkörüli vezetéknélküli kommunikációs rendszer)

HTTP – Hyper Text Transfer Protocol (hipertext átviteli protokoll)

HTTPS – HTTP Secure (biztonságos HTTP)

I/O – input/output (bemenet/kimenet)

IM – Instant Messaging (közvetlen üzenetváltás)

MILNET – Military Network

MMS – Multimedia Messaging Service (multimédiás üzenetküldési szolgáltatás)

NFSNet – National Science Foundation Network

PDF – Portable Document Format (hordozható dokumentum formátum)

PHP – PHP: Hypertext Preprocessor

QoS – Quality of Service (szolgáltatás minősége)

RS – Revenue Sharing (bevételmegosztás)

SFTP – Secure FTP (biztonságos FTP)

SMS – Short Messaging Service (rövid üzenetküldési szolgáltatás)

SQL - Structured Query Language (strukturált lekérdezőnyelv)

SSH – Secure Shell (titkosított távoli konzolkapcsolat)

UI – User Interface (felhasználói felület)

UMTS – Universal Mobile Telecommunications System (univerzális vezetéknélküli telefon szolgáltatás)

URL – Uniform Resource Locator (egységes erőforráshely meghatározó)

VPN – Virtual Private Network (virtuális magánhálózat)

WAP – Wireless Application Protocol (vezetéknélküli alkalmazások protokollja)

WWW – World Wide Web (világháló)

XLS – Excel spreadsheet (a Microsoft Excel táblázatkezelő formátuma)

IX. Köszönetnyilvánítás

Először is konzulensemnek, Szőllősi Lorándnak szeretném megköszönni az időt és energiát, amit a munkám támogatásának szentelt.

Köszönöm szüleimnek, Markert Károlynak és Markertné Oláh Erikának azt, hogy tökéletes körülményeket teremtettek tanulmányaimhoz. E diplomadolgozatot édesapámnak ajánlom, aki 1960-ban diplomázott ugyanezen a karon.

Legjobb barátomnak, Tóth Gergelynek köszönöm, hogy az elmúlt öt év során mindig, mindenben számíthattam rá. Remélem, hogy barátságunk az egyetem elvégzését követően is megmarad!

Köszönöm továbbá Tóth Bálint, Balatoni Emese, Ertli Gergő, Bacsárdi László és Kereskényi Balázs segítségét, amivel e dolgozat megírásához hozzájárultak.

Forján Reginának pedig azért jár a köszönet, mert májusban – amikor a legnagyobb szükségem volt a támogatásra – mellettem volt. És remélem, hogy ez még nagyon sokáig így is marad…