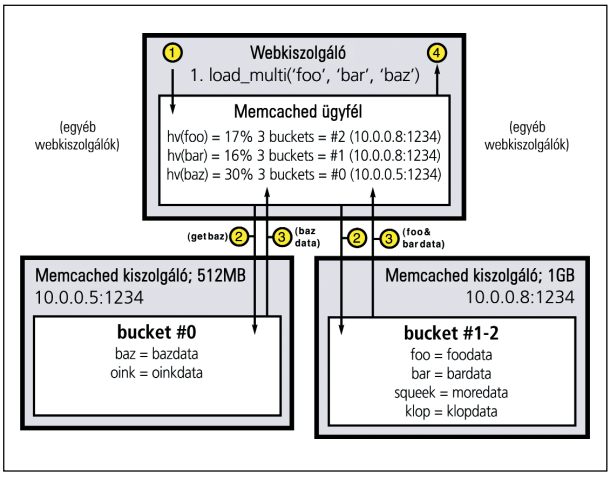
Elosztott gyorstárazás a Memcached használatával

Webkiszolgálónk adatbázisának terhelését jelentősen csökkenthetjük, ha alkalmazásainkat egy méretezhető, objektumokat gyorstárazó réteggel bővítjük.

A ***Memcached*** nagy teljesítményű, elosztott gyorstárrendszer. Bár a konkrét alkalmazásoktól tökéletesen független, leginkább a webes alkalmazások gyorsítására használják úgy, hogy segítségével az adatbázisok terhelését csökkentik. A ***Memcachedet*** többek közt a ***LiveJournal***, a ***Slashdot*** és a ***Wikipedia*** is használja.

Miért gyorstárazunk?

Az elmúlt nyolc év során nagy méretű, interaktív, adatbázisra épülő, több kiszolgálóra is kiterjedő weboldalak készítésével foglalkoztam. A ➲ ***LiveJournal.com*** blogrendszeréhez és nyilvános hálózati szolgáltatásaihoz közel 70 gép biztosítja a hátteret, összesen 2,5 millió fiókot tartva fenn. A szokásos blogolási lehetőségeken és a barátok/érdeklődés/személyes adatok megadásának lehetőségén túl a ***LiveJournal*** oldalán fórumokat, kérdőíveket, a felhasználók egyedi igényeire szabott híroldalakat, telefonokról áttöltött hangfelvételeket és egyéb változatos, az embereket egymáshoz közelebb hozó dolgokat és szolgáltatásokat lehet találni. A dinamikus weboldalak előállítási sebességének növelése mindig nehéz feladat, és ez alól a ***LiveJournal*** sem volt kivétel. A dolgot tovább nehezítette, hogy a rendszernek szinte minden tartalmi eleméhez tartozhat valamilyen biztonsági szint, illetve az elemek számos különböző nézetbe csoportosíthatók. Mivel korábban már foglalkoztam dinamikus, környezetfüggő tartalmakkal, tudtam, hogy statikus oldalak előzetes előállításával nem sokra jutunk. Az oldalakat alkotó különféle objektumok gyorstárazási jellemzői és élettartamuk annyira eltérő lehet, hogy az oldalakat többször kellene előállítani, mint ahányszor lekérik őket. Így ezzel a megoldással csak az erőforrásokat pazaroltuk volna.
Persze a gyorstárazás ettől még nem rossz dolog. A másik oldalról viszont egy számítógép sebességét jelentős mértékben befolyásolja memóriarendszerének sebessége, mérete és szintjeinek száma. A gyorstárazás valóban hasznos, de csak akkor, ha megfelelő tárolóeszközzel és felbontással tudjuk végezni. Én úgy gondolom, a legjobb az, ha az egyes oldalak objektumait külön gyorstárazzuk, és nem magukat a weboldalakat tároljuk. Ilyenkor nem növeljük feleslegesen a szükséges tárterület méretét azzal, hogy az objektumokat és a több oldalon is feltűnő sablonelemeket minden oldalhoz külön tároljuk. A végső döntés persze mindig különféle szempontok között egyensúlyozva születik meg. A processzorok egyre gyorsabbak, tehát szerintem inkább a processzor erőforrásait érdemes használni, és nem a lemezeket. Igaz ugyan, hogy az újabb lemezek egyre nagyobbak és olcsóbbak, a sebességük azonban korántsem nő ilyen mértékben. Figyelembe véve, hogy mennyire hajlamosak a meghibásodásra, és mennyire tohonyák, az a legjobb, ha nem is használjuk őket, ha nem muszáj.
A ***LiveJournal*** webes csomópontjaiban egyáltalán nincsenek merevlemezek. A rendszerindítást közös, redundáns módon tárolt NFS képről hajtják végre. Ez a megoldás nemcsak olcsóbb, de kevesebb karbantartást is igényel.
Az adatbázis-kiszolgálóból természetesen nem spórolhatjuk ki a lemezeket, itt viszont gyors, RAID-be kötött meghajtókat használunk. Jelenleg tíz különböző adatbázisfürtöt tartunk fenn, melyek mindegyike kettő vagy több gépből áll. A tíz fürtből kilenc felhasználói, vagyis ezek kizárólag a felhasználók adatait tárolják. A tizedik egy közös fürt, amely a nem a felhasználókkal kapcsolatos adatokat hordozza. Itt találhatók a felhasználó-fürt hozzárendelések is. A fürtöket az írási műveletek elosztása végett függetlenítettük egymástól. Megtehettük volna azt is, hogy egyetlen nagy méretű fürtöt állítunk össze, több száz szolgagéppel, ez azonban csak az olvasásokat tudta volna szétosztani. A válaszidő



1. ábra A Memcached ügyfélkönyvtár feladata a kérések megfelelő kiszolgálókhoz való továbbítása

minden újabb szolga hozzáadásával növekedett volna, a kiírt adatok elkerülhetetlen frissítése miatt egyre rosszabb teljesítményt kaptunk volna.
Az elmondottakból talán már látszik is, hogy a ***LiveJournal*** háttérrendszerét milyen elgondolás szerint építettük meg:

1. Minél kevesebb lemezt használunk, mert csak a baj van velük. Ha elkerülhetetlen a használatuk, akkor csak gyors és redundáns be- és kiviteli alrendszert veszünk igénybe.
2. Szélességében méretezzük a rendszert és nem felfelé, vagyis inkább több kis gépet üzemeltetünk, mint kevesebb nagyot.

A kis gép persze nem az olcsó, inkább a sokáig használható gépet jelenti. Én szeretek addig megtartani egy gépet, ameddig az általa elfoglalt hely és a megtermelt hő szempontjából érdemes. Méretezésnek nevezik azt is, amikor félévente kidobják a gépet, és vesznek egy még nagyobbat, de ez a megoldás nekem nem nagyon tetszik.

Hol gyorstárazzunk?

A Memcached bevezetése előtt a webes csomópontok kedvük és igényük szerint érhették el az adatbázisokat. A rendszer működött, de nem volt az igazi. Hamar rájöttem, hogy hiába van 4 vagy 8 GB memória az adatbázis-kiszolgálóban, gyorstárai korlátozottak, mind magának a memóriának a méretét, mind a 32 bites gépeken futó folyamatok által megcímezhető terület nagyságát tekintve. Persze vehettem volna 64 bites gépeket, amelyek több memóriát tudnak megcímezni, de mit tegyek, makacs és fösvény vagyok. Úgy döntöttem tehát, hogy a webes csomópontokon kell többet gyorstárazni. Mivel ***mod_perl 1.x***-et használunk, a gyorstárazás egyszerűen kínszenvedés. Minden folyamat, vagyis minden webes kérés saját címtérrel rendelkezik, a többivel nem tud adatokat megosztani. Meg lehetne csinálni, hogy a 30-50 folyamat mindegyike saját gyorstárat tartson fenn, de megint csak pazarolnánk az erőforrásokat. A System V megosztott memória megoldása ma már meglehetősen szokatlan korlátokkal küzd, nem hordozható, és csak egyetlen gépen működik. Több mint 40 csomóponton egyszerűen nem is használható. Azt hiszem, érzékelhető, hogy mennyire nem voltam elégedett a már meglévő gyorstárazó megoldásokkal. Még ha alkalmazásrendszerünk többszálú is lett volna, és az adatokat elvileg könnyen meg lehetett volna osztani a folyamatok között, ez akkor is csak egy-egy gépre terjedt volna ki. El akartuk kerülni, hogy a több mint 40 gép önállóan gyorstárazzon, és ki tudja, hány példányt tartson fenn az adatokból.

A Memcached születése

Egy napon, amikor már kellően tele volt a puttonyom a ***mod_perl*** alkalmazások gyenge gyorstárazásával, álmodozni kezdtem. Rájöttem, hogy egy csomó szabad memória található elszórva a hálózaton, és valahogy ki kellene használni. Aki ***Perl*** programozóként turkál egy kicsit a ***CPAN***-on, ***Cache::**** modulok ijesztő sokaságát találja. Legtöbbjük felülete valamilyen szótár. Akinek ez a kifejezés nem mond túl sokat, annak elárulom, hogy a szótár olyan elvont adattípus, amely kulcsokat és értékeket rendel össze. A Perlben járatos emberek az ilyesmit asszociatív tömbnek vagy kivonattáblának, esetleg röviden kivonatnak nevezik. A kivonattábla különleges típusú adatszerkezet, amely egy szótárfelületet bocsát rendelkezésünkre.
Olyan átfogó kivonattáblát akartam létrehozni, amelyet az összes gép összes webes folyamata akár egyszerre is el tud érni, és azonnal látja a többiek módosításait. Ez lett volna a gyorstáram. Mivel a memória drága, a hálózatok gyorsak, a kiszolgálók üzembiztossága pedig kérdéses, a gyorstárat az összes gépre el akartam osztani. Gyors keresést végeztem, és mivel ilyet nem találtam, nekiláttam megvalósítani az elképzelésemet.
A ***Memcached*** kiszolgáló példányok mindegyike egy felhasználó által megadott IP-n és kapun várja a kéréseket. Az alapötlet az, hogy a hálózaton mindenfelé, ahol csak szabad memóriát találunk, ***Memcached*** példányokat futtatunk, és ezek szolgálják ki az alkalmazásokat. Egy-egy gépen akár több példányt is lehet futtatni, ha az adott gép 32 bites, és több memóriával rendelkezik, mint amit a rendszermag egy-egy folyamat számára biztosítani tud. Például, mielőtt még rájöttünk volna, hogy inkább szélességében, mint felfelé érdemes méretezni a rendszert, vettünk néhány elképesztően drága, egyenként 12 GB memóriával felszerelt gépet. Mostanában mindenféle feladatokra használjuk őket, többek közt öt-öt 2 GB-os ***Memcached*** példányt futtatnak. Így egyetlen gép is 10 GB memóriával képes hozzájárulni az átfogó gyorstárhoz. Mindezt annak ellenére, hogy 32 bites Linux alatt egy-egy folyamat általában csak 3 GB memóriát tud megcímezni.
A ***Memcached*** trükkje az, hogy adott kulcs kezelésére mindig ugyanazt a csomópontot kell felkérni, miközben a tároló kulcsok egyenletesen oszlanak el a csomópontok között. Arra nincs lehetőség, hogy a „példa” kulcsot az 1-es gépen tároljuk, majd egy másik folyamat később a 2-es gépről próbálja betölteni. Persze ez nem is okoz különösebb gondot. A ***Memcached*** csomópontokat úgy érdemes kezelni, mint a kivonattáblák vödreit.

A Memcached működése

1. lépés: Az alkalmazás a példa1, a példa2 és a példa3 iránt bocsát ki kérést az ügyfélkönyvtáron keresztül, ez kiszámítja a kivonatok értékét, amiből ki tudja választani, hogy melyik ***Memcached*** kiszolgálóhoz kell fordulnia.
2. lépés: A ***Memcached*** ügyfél párhuzamos kéréseket küld a megfelelő ***Memcached-*** kiszolgálóknak.
3. lépés: ***Memcached*** kiszolgálók válaszolnak az ügyfélkönyvtárnak.
4. lépés: A ***Memcached*** ügyfélkönyvtár összegyűjti az alkalmazásnak szóló válaszokat.

Aki ismeri a kivonattáblák működését, nyugodtan ugorjon. Aki nem, annak azonnal adok egy gyors áttekintést. A kivonattábla vödrök tömbje. Minden vödör (azaz tömbelem) csomópontok egy listáját tartalmazza, minden csomópont egy kulcs-érték párosból áll. A megfelelő kulcsot tartalmazó csomópont után szükség esetén ebben a listában végzünk keresést. A legtöbb kivonattábla először kisméretű, majd idővel dinamikusan változik a mérete, ahogy a vödrökben tárolt listák túl hosszúakra nyúlnak.
Ha adott kulcshoz tartozó értékre vonatkozóan olvasási vagy írási kérést indítunk, akkor a kulcs alapján először egy

kivonatot kell számolni. A kivonatoló függvények egyirányú leképező függvények, amelyek egy kulcshoz (ez szám vagy karakterlánc egyaránt lehet) valamilyen számot rendelnek – ez lesz a vödör száma. A vödör számának meghatározása után keresést kell végezni a vödör listájában az adott kulcsú csomópont után. Ha nincs ilyen, akkor új csomóponttal kell bővíteni a listát.

Mi köze van mindennek a ***Memcached*** működéséhez? A ***Memcached*** a felhasználók számára egy szótárfelületet (kulcs -> érték) biztosít, de belül két rétegű kivonatolást végez. Az első réteg az ügyfélkönyvtárban található. A kulcsot képzetes vödrök listájába kivonatolva ez dönti el, hogy melyik ***Memcached*** kiszolgálónak kell elküldeni a kérést. Amikor a kérés eljutott hozzá, a kiválasztott ***Memcached*** kiszolgáló normál kivonattáblát alkalmaz.

Minden ***Memcached*** példány tökéletesen független, a többivel semmilyen adatcserét nem folytat. Mindegyik példány a legrégebben használt elemeket dobja el, amikor helyet teremt az újaknak. A kiszolgálók rengeteg statisztikát készítenek, amelyek alapján összesíthető a teljes ***Memcached*** telep lekérdezés/találat/hiba mutatója. Ha valamelyik kiszolgáló meghibásodna, az ügyfelek képesek a halott gép vagy gépek megkerülésére, a továbbra is aktív kiszolgálókat pedig tovább használják. Ez a szolgáltatás elhagyható, ugyanis használatához az alkalmazásokat is fel kell készíteni arra, hogy esetleg elavult adatokat kapnak valamelyik betegeskedő csomóponttól. Ha a szolgáltatást kikapcsoljuk, akkor a halott kiszolgálókhoz intézett kérések egyszerűen egy gyorstárhibát eredményeznek az alkalmazás oldalán. Ha kellően nagy, elegendő egyedi állomásból álló ***Memcached*** telepet tartunk fenn, akkor egy-egy gép leállása nem fogja számottevő mértékben rontani az átfogó találati arányt.

Saját rendszerünk felépítése

A ***LiveJournal.com*** jelenleg 28 ***Memcached*** példányt futtat összesen tíz gépen, ezek a leggyakrabban igényelt 30 GB-nyi adatot gyorstárazzák. Találati arányunk 92 % körül mozog, vagyis az alkalmazásoknak a korábbinál jóval kevesebbszer kell az adatbázisokhoz fordulniuk.

Webes csomópontjaink mindegyike 4 GB memóriával rendelkezik, ezeken három-három 1 GB-os ***Memcached*** példányt futtatunk, a mod_perl 500 MB-tal gazdálkodhat, 500 MB pedig tartalékként szolgál. A ***Memcached*** és a ***mod_perl*** azonos gépen való futtatása jó megoldás, a ***mod_perl*** kódunk ugyanis erősen terheli a processzort, a ***Memcached*** viszont csak alig. Természetesen megtehetnénk, hogy a ***Memcached*** futtatására külön gépeket vásárlunk, de gazdaságosabb megoldás a ***Memcached*** példányokat oda szétszórni, ahol amúgy is rendelkezünk felhasználható memóriával, esetleg memóriabővítést végezni egy régebbi gépen, ami így meg fog felelni a feladatra.

Memcached telepet különféle méretű gyorstárakkal is lehet futtatni, mi például 512 MB, 1 GB és 2 GB méretű példányokat keverünk. A példányokat és méretüket az ügyfélbeállítások között kell megadni, a ***Memcached*** kapcsolatobjektumok ennek megfelelően súlyoznak.

Sebesség

A gyorstárazás elsődleges célja nyilván a folyamatok gyorsítása, tehát a ***Memcached*** rendszert a lehető leggyorsabbra készítettük. A program első, kísérleti példányát Perlben írtuk. Szeretem a Perlt, de a kísérleti kód nevetségesen lassú és nagyméretű volt. A ***Perl*** meglehetősen sok memóriát használ, és nagyobb számú hálózati kapcsolatot sem képes egyszerre kezelni.

A jelenlegi változat C nyelven készült, egy folyamatból álló, egyszálú, aszinkron be- és kivitelt végző, esemény alapú démon. A hordozhatóság és a megfelelő sebesség érdekében az eseményekről való értesítéseket a ***libevent*** segítségével oldottuk meg. A ***libevent*** használata azért előnyös, mert futási időben a lehető legjobb módszert választja a fájlleírók kezelésére. BSD alatt például a ***kqueue***, Linux 2.6 alatt pedig az ***epoll*** szolgáltatásaira épül, amelyek több ezer párhuzamos kapcsolat hatékony kezelésére is megfelelők. Más rendszereken a ***libevent*** a hagyományos ***poll*** és ***select*** eljárásokat használja.

A ***Memcached*** belsejében minden algoritmus ***O(1)*** futási idejű. Az algoritmusok végrehajtási ideje és processzorigénye tehát az egyszerre csatlakozó ügyfelek számától – legalábbis a ***kqueue*** vagy az ***epoll*** használatakor – független, ahogy az adatok méretétől és az egyéb tényezőktől is.

Megjegyezném, a ***Memcached*** tömbfoglalót (***slab allocator***) használ a memóriafoglalásokhoz. A ***Memcached*** korai változatai a ***glibc***-féle ***mallocot*** használták, ám ez nagyjából egy heti működés után fura dolgokat kezdtek produkálni, és a címtér széttöredezése miatt csak a processzor erőforrásait fogyasztotta. A tömbfoglalók csak nagy méretű memóriaterületeket foglalnak le, ezeket a különféle elemosztályok számára kisebb részekre osztják, majd egy részek foglaltságát nyilvántartó lista vezetésével figyelik, hogy az egyes osztályok objektumai mikor szabadulnak fel. Erről a témáról további részleteket a források között szereplő Bonwick-féle leírásban lehet találni. A ***Memcached*** pillanatnyilag 64 bájt és 1 MB között képes területet foglalni úgy, hogy a memória nagysága kettő hatványa legyen. Minden objektumot a befogadására alkalmas legkisebb egységben helyez el. A tömbfoglaló használatának köszönhetően a megfelelő teljesítményt tetszőleges időtartamra biztosítani tudjuk. Az éles ***Memcached*** kiszolgálóink például 4-5 hónapon keresztül üzemeltek, átlagosan 7000 lekérdezést kezeltek másodpercenként, de semmilyen gond nem volt velük, és processzorterhelésük is alacsony maradt.

A ***Memcacheddel*** szemben további elvárás volt, hogy képes legyen zárolások nélkül működni. Minden objektummal kapcsolatban belső változatkövetést és hivatkozásszámlálást végez, így egyik ügyfél sem akadályozhatja egy másik műveleteit. Előfordulhat például, hogy egy a csomagok felét eldobáló rossz hálózati kapcsolattal rendelkező ügyfél éppen frissít egy olyan a objektumot, amit közben több tucat másik ügyfél tölt le. Még ekkor sem kell senkinek sem bárki másra várakoznia.

A működésről még annyit megjegyeznék, hogy a protokoll egyszerre több kulcs lekérdezését is lehetővé teszi, ami főként akkor hasznos, ha egy alkalmazás előre tudja, hogy több száz kulcsot is elő kell keresnie. Ilyenkor nem soros lekérdezést indít, ami a hálózati válaszidők miatt viszonylag sokáig tartana, hanem egyetlen kéréssel elintézi az összeset. Ha szükséges, az ügyfélkönyvtárak önműködően több, a ***Memcached*** példányokhoz egymással párhuzamosan indított többkulcsos lekérdezésre osztják az alkalmazásoktól ér-

kező többkulcsos kéréseket. Arra is van lehetőség, hogy az alkalmazások előre meghatározott kivonatértékeket csatoljanak az adatokhoz, így egy-egy adatcsoportot azonos példányon tudnak tartani. Ezzel a megoldással az ügyfélkönyvtár némi processzoridőt is megtakaríthat, hiszen a kivonatokat nem kell kezelnie.

Az ügyfélkönyvtárak

A ***Memcached*** ügyfél/kiszolgáló felülete egyszerű, könnyen áttekinthető. Ügyfélkönyvtár Perl, PHP, Python és Java alá egyaránt létezik. Úgy hallom, egyik munkatársam egy Ruby ügyfélen is dolgozik, ami hamarosan meg fog jelenni.
Az ügyfelek mindegyike támogatja az objektumok sorrendezését, erre saját, natív sorrendező eljárásukat használják: a ***Perl*** a ***Storable***, a ***PHP*** a ***serialize***, a ***Python*** a ***Pickle***, a ***Java*** pedig a ***Serializable*** felületet veszi igénybe. A legtöbb ügyfél képes az átlátszó tömörítés támogatására, amelyet beállítástól függően csak adott adatméret felett használ. A sorrendezés és a tömörítés használata azért vált lehetségessé, mert a ***Memcached*** megengedi az ügyfeleknek, hogy átlátszatlan jelzőket – ezek a bejövő adatok kezelésének módját határozzák meg – tároljanak az egyes tételek mellett.

A Memcached használata

A ***Memcached*** telepítése önmagában nem bonyolult, használata viszont már komolyabb körültekintést igényel. Azt, hogy a legtöbb idő milyen műveletekkel telik el, csak saját alkalmazásaink vizsgálatával tudjuk eldönteni – ezeken a helyeken kell aztán gyorstárazni. A gyorstár frissítéséről és tisztán tartásáról szintén gondoskodnunk kell, a gyorstár tartalmának egységessége ugyanis a legtöbb alkalmazás számára fontos. Ha alkalmazásunk belső API-ja már letisztult, és az adatbázist nem ide-oda kapkodva érjük el, akkor a ***Memcached*** támogatásának hozzáadásával nem lesz gond. A lekérdező függvényekben egyszerűen csak ***Memcached***-hez kell elsőként fordulni. Hiba esetén lépjünk tovább az adatbázishoz, majd töltsük fel a ***Memcached*** tárait is. Az adatírásokat végző függvényekben az adatbázis és a ***Memcached*** frissítését egyaránt végezzük el. Minden bizonnyal lesznek olyan versenyhelyzetek és adategységességi gondok, amelyek megoldásra várnak, ám a ***Memcached*** API-ja biztosítja a kezelésükhöz szükséges eszközöket.
A ***Memcached*** olyan adatok tárolására is használható, amelyeket lemezre nem akarunk kiírni. A ***LiveJournal*** például úgy előzi meg az adatküldések véletlen kettőződését, hogy a tranzakciók kódjait a ***Memcached*** segítségével tárolja, kulcsként a tranzakciók aláírását használva. A ***Memcached*** elsődleges adattárként való használatára egy másik példa az egyszerű és/vagy rosszindulatú robotok és szemétküldők kihajigálása. Ha figyelemmel követjük, hogy az egyes IP-címekről származó kapcsolatok révén milyen műveleteket és mikor indítottak, akkor bizonyos minták alapján már korán fel tudjuk ismerni a támadásokat, és meg tudjuk kezdeni a szükséges intézkedéseket. Ezeket az adatokat felesleges volna adatbázisban tárolni, csak a lemezmeghajtókat terhelnénk velük. Bár a memóriában való tárolás elegáns megoldás, ha valamelyik ***Memcached*** csomópont meghibásodik, akkor az adatok is elvesznek.
A levelezőlistán megkérdeztem, hogy mások mire használják a ***Memcached***-et, és a következő válaszokat kaptam:

- Sokan alkalmazzák ugyanarra a célra, mint a ***LiveJournal***, vagyis kisméretű webes objektumok gyorstárazására.
- Az egyik oldalon az éppen lejátszás alatt lévő zeneszámot adják át vele Java alapú adatfolyam-kiszolgálójuktól PHP alapú webhelyüknek. Erre a célra korábban egy adatbázist használtak, de a ***Memcached*** szerintük jobban működik.
- Sokan hitelesítési adatokat és munkamenetkulcsokat gyorstáraznak vele.
- Az egyik helyen az ismert jó és rossz állomások és hitelesítési adataik gyorstárazásával a levélkiszolgálók működését gyorsítják.

Mindig újabb és újabb érdekes elektronikus leveleket és javaslatokat kapok – örömmel látom, hogy sokaknak tudtunk segíteni.

Egyéb megoldások

Ha valaki egy többszálú alkalmazást egyetlen gépen futtatva meg tudja oldani feladatait, illetve nincs szüksége átfogó gyorstárra, annak nincs mit kezdeni a ***Memcached***-del. Aki csak egy gépet használ, annak a ***SysV*** megosztott memória szolgáltatása is megfelelő lehet.
Néhányan felvetették, hogy a ***MySQL 4.x*** lekérdezés-gyorstárának alkalmazásával a ***Memcached*** kiiktatható. A ***MySQL*** lekérdezés-gyorstáblája minden alkalommal kiürítésre kerül, amikor valamelyik érintett tábla – bármilyen módon is – frissítésre kerül, ezért inkább olyan helyeken használható, ahol a tartalmat csak olvassák. A ***LiveJournal***-on rengeteg írás történik, ahogy a nagy forgalmú webhelyeken általában lenni szokott. Más adatbázisokhoz hasonlóan a MySQL gyorstárainak mérete összesen nem haladhatja meg a rendszermag által biztosított címteret, ami a 32 bites gépeken 3 GB – manapság ez már kevésnek számít.
Sokaknak megfelelő megoldás a ***MySQL*** memóriabeli táblakezelője is. Nekem azért nem tetszett, mert csak szabott méretű rekordokkal tud dolgozni, ***BLOB*** és ***TEXT*** oszlopok használatát nem engedi. Az általa tárolható adatok összesített mérete is korlátozott, vagyis belőle is nagy számú példányt kellett volna futtatni, és ugyanúgy gondoskodni kellett volna a kulcsok szétosztásáról.

Köszönetnyilvánítás

Szeretnék köszönetet mondani ***Anatoly Vorobey*** fejlesztőnek, ***Lisa Phillipsnek***, amiért hajlandó volt a korai, folyton összeomló változatokkal küzdeni, továbbá a levelezőlista tagjainak, akik rengeteg foltot, kérdést és javaslatot küldtek nekünk.

Linux Journal 2004. augusztus, 124. szám

Brad Fitzpatrick nyolc évig adatbázis alapú webhelyek építésével foglalkozott. Brad szeret kerékpározni, illetve különféle, egyébként csak pénzköltéssel megoldható gondokra alternatív megoldásokat kitalálni. Aki nem kék tablettákat forgalmaz, és nem halott kiszolgálókról akar értesítést küldeni, annak levelét örömmel fogadja a brad@danga.com címen.