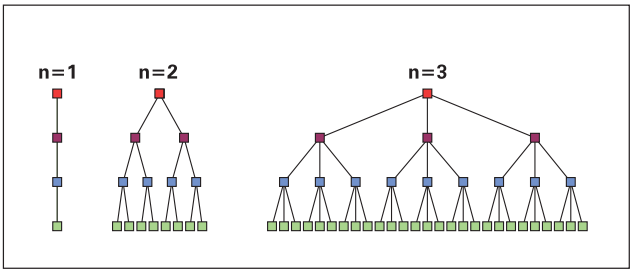
Reiser4: hatékonyan gyorstárazó fák tervezése (2. rész)

A Reiser fájlrendszer negyedik változata a nagyobb teljesítmény érdekében jobban ellapítja a faadatszerkezetet, mint a harmadik változat tette. A leírtakból megtudhatjuk, hogyan is fest ez a gyakorlatban.

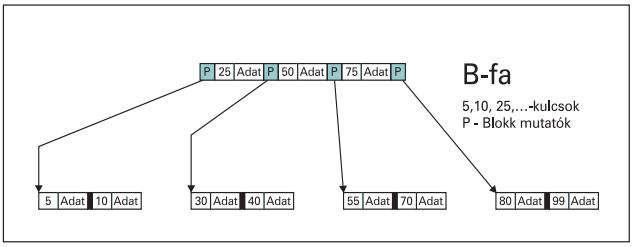
Cikkünk a Reiser fájlrendszer felépítéséről szóló cikksorozatunk második része. Az első cikkben (Linuxvilág, 2003. március) az alapokat tárgyaltuk: a fákat, a csomópontokat és a cikkelyeket. Ebből a cikkből azt tudhatod meg, hogy miért jobbak a kiegyensúlyozott fák a kiegyensúlyozatlan fáknál, és miért jobbak a B+ fák a B-fáknál, miközben a gyorstárazási alapelveket is megismered. Ezt követően arra is fény derül, hogyan alkalmazza ezeket az említett elemeket a Reiser fájlrendszer az adatbázis-rendszereknél már megismert BLOB-ok, vagyis nagyméretű bináris egységek esetében. Mindez azt sugallja, hogy a BLOB-ok csökkentik a belső csomópontok gyorstárazásának a hatékonyságát, mivel a fa így kiegyensúlyozatlanná válik. Azt is megtudjuk, hogyan tárol a Reiser fájlrendszer olyan szerkezeteket, amelyek nagyobb méretűek egy csomópontnál, téve ezt anélkül, hogy a fa kiegyensúlyozatlanná válna.
Elnézést az olvasóktól, hogy mindenkit ennyire megvárattam ezzel a cikkel – a késedelem oka a Halloweenkor esedékes 2.6 miatti bővítésbefagyasztás volt, mivel akkortájt kellett a Reiser4-et gyorsan megbízható formába hozni.

Elágaztatás

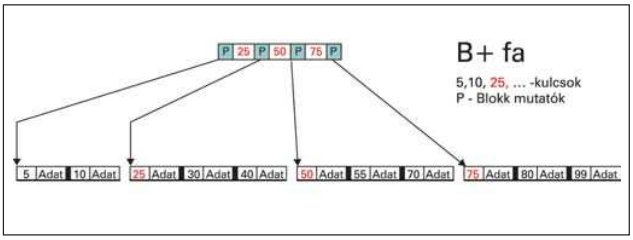
Az elágaztatási ütem (n) a csomópontok számát jelöli, amelyekre az egyes szintek csomópontjai mutatnak (*1. ábra*). Ha minden egyes csomópont az alatta lévő szinten `n` csomópontra mutathat, akkor a tetejéről elindulva a gyökércsomópont a következő szinten `n` számú belső csomópontra mutat, amelyek mindegyike `n` számú belső csomópontra mutat az azt követő szinten és így tovább. Az `m` szintű belső csomópont `nm` levélcsomópontra tartalmazhat hivatkozást, amelyek az utolsó szinten rendelkeznek elemekkel. Minél több adatot próbálsz tárolni egy fában, annál nagyobb mezőkre van szükséged a kulcsoknál, amikkel különbséget lehet tenni az elemek között, majd pedig az adott elem valamelyik része (egy bizonyos eltolásnál) kijelölhető. Ez azt jelenti, hogy a kulcsaidnak nagyobbaknak kell lenniük, ennek következtében nő az elágazási szám (kivéve, ha a kulcsokat tömöríted, de ezzel a következő változat megjelenéséig még várni kell).
Az 1. ábra első része egy négyszintű fát ábrázol, aminek az elágazási száma n = 1. Csupán négy darab csomóponttal bír, ezek a vörös gyökércsomóponttal indulnak, átlépve a sötétvörös, vagyis belső csomópontba, majd a kék gallycsomópontba, végül pedig a zöld levélcsomópontba, ami a tényleges adatot tartalmazza.
A második négyszintű fa (amelynek az elágazási száma n = 2) egy gyökércsomóponttal kezdődik, amit két belső csomópont követ, amelyek mindegyike két gallycsomópontra mutat (ez összesen négy gallycsomópontot jelent), és mindegyikük tovább egy levélcsomópontba, ami összesen nyolc levélcsomópontot jelent. Végül pedig egy négyszintű, n = 3 elágazási számú fát láthatunk, melynek egy gyökércsomópontja, három belső csomópontja, 9 gallycsomópontja, és 27 levélcsomópontja van.



1. ábra Három darab négyszintű kiegyensúlyozott fa



2. ábra Egy B-fa



3. ábra Egy B+fa

A B+ fák jobbak a B-fáknál

A belső csomópontokban nemcsak mutatókat és kulcsokat lehet tárolni, hanem a kulcsokhoz tartozó elemeket is. Erre volt képes az eredeti B-fa algoritmus (*2. ábra*). Ezt követően találták ki a B+ fákat, amelyek csak mutatókat és kulcsokat tárolnak a belső csomópontokban, a kapcsolódó elemek pedig levélszinten tárolódnak (*3. ábra*).

A B+ fák elágaztatási jobb a B-fákénál

Az elágaztatás növelhető, ha a belső csomópontokban csak mutatókat és kulcsokat helyezünk el, továbbá nem hígítunk az elemek adataival. A nagyobb elágaztatásnak köszönhetően a belső csomópontok hatékonyabban gyorstárazhatók, mivel így kevesebb belső csomópontot kell kezelni. Az emberek erre gyakran azt mondják, hogy „de a B-fák elemeket gyorstáraznak, és az effajta gyorstárazás éppoly hasznos". Általánosságban azonban nem ez a helyes válasz. Ha az általánosságot

vesszük alapul, a vita eldöntése még nehezebb lehet. De mielőtt erre jobban kitérnénk, tekintsük át a gyorstártervezés alapjait! Tételezzük fel a következőket:

- Kétcsoportnyi elemmel rendelkezel: A-val és B-vel.
- Tegyük fel, hogy a két csoportból változó rendszerességgel szükségesek az elemek: bizonyos dolgokra gyakrabban van szükség, másokra ritkábban, és az egyes csoportba tartozó elemek idővel változhatnak.
- Bizonyos elemeket kéznél tartunk, miután a korlátozott méretű gyorstárból előszedtük őket.
- Az A csoportban található elemekhez olykor egy B csoportbeli elemet kell kötni, ami annyit tesz, hogyha szükségünk van egy elemre az A csoportból, akkor annak a B csoportbeli társára is szükség lesz.

Így nő az A csoportból a gyakran használt elemek tárolásához szükséges gyorstár mérete. Ha erős kötődés van két szükséges elem között, amelyek a csoportok egyikében egymáshoz kötöttek, és ez a kötődés erősebb, mint a gyorstárazáshoz felhasznált erősforrásokon elért nyereség, ahol A és B az LRU (Least Recently used: újabban legkevésbé használt) algoritmusnak megfelelően gyorstárazódik, az hasznos lehet. Ha azonban nincs efféle kapcsolat, akkor kifejezetten rossz. Az LRU egy olyan algoritmus, amely az újabban kevésbé használt elemeket szórja ki a gyorstárból, így szabadít fel helyet. Az LRU különféle változatai az operációs rendszerek fejlesztésekor használt leggyakoribb gyorstárazási algoritmusok közé tartoznak.
De várjunk csak! Mi történik akkor, ha a B csoportból is szükséged van valamilyen elemre, s ezért jó lenne, ha innen is néhány elem a gyorstárba kerülne, tehát a B csoport valamelyik szeletére lenne szükség? Ebben az esetben az a gond, hogy kapcsolat nélkül nem valószínű, hogy neked a B csoportból arra az elemre lesz szükséged, ami valamelyik A-beli elemhez kötődik. Ha kiválasztasz valamit a B-ből, akkor az a gyorstárban tárolódik, és ezáltal az LRU-ra épülő gyorstár hatékonysága csökken, kivéve, ha a gyorstárazásért egy másik, legalább olyan jó algoritmus felel, mint az LRU. Amikor azt próbáljuk kiválasztani, hogy az A csoportnak megfelelően a B csoportból melyik elem gyorstárazása lenne ajánlatos, gyakran megesik, hogy az adott algoritmus nem olyan jó, mint az LRU, így megint bajban vagyunk.
A dolgoknak kevésbé hatékony összekötése, amikre rendszertelenül van szükségünk, a számítástechnikai világon kívül gyakori. Például tegyük fel, hogy szereted a pattogatott kukoricát és a szusit, de csak bizonyos napokon eszed őket és teljesen véletlenszerűen. Tételezzük fel, hogy a megnézendő filmeket is teljesen véletlenszerűen választod ki. Vegyük úgy, hogy a mozi megköveteli, hogy az adott, véletlenszerűen kiválasztott film esetében csakis pattogatott kukoricát ehetsz, és nem ehetsz szusit, amit pedig a sarki étteremben megkapnál. Ez lenne a társadalmilag legelőnyösebb rendszer? Tegyük fel, hogy a hot dogok minősége az egyes hotdog-árusok között véletlenszerűen változik. Ha a legjobb moziban egy bizonyos este csak a moziban készült hot dogot eheted, és a külső árusnál vásárolt hot dogot nem viheted be a mozi területére, az szerinted a társadalmilag legelőnyösebb? Előnyös-e ez számodra? A szorosan egymáshoz kapcsolódó dolgok összerendelése néha a teljesítmény szempontjából is előnyös. Számos fájlrendszer összerendeli a fájlméretadatokat a fájlnévadatokkal, ami úgy tűnik, hogy jól működik, legalábbis jobban, mint ahogyan az LRU működne.
Gyakori hiba a gyorstárak tervezésekor, hogy olyan dolgokat kötünk egymáshoz, amelyek egyébként nem állnak kapcsolatban, de ez még nem elég ahhoz, hogy megértsük, miért jobbak a B+ fák a többinél. Belső csomópontok esetében egyetlen csomópontban több mutatót is tárolunk, ezáltal a mutatók nem külön-külön kerülnek a gyorstárba. Vitatható, hogy a mutatók és az általuk hivatkozott elemek szorosabban összefüggenek-e, mint több különböző mutató. Remélem, a leírtak valóban tanulságosnak bizonyulnak, azonban még mindig meg kell értenünk egy gyorstárazási alapelvet.

Saját meghatározásom a gyorstár hőmérsékletére

Legyen a gyorstár hőmérséklete egy olyan érték, ami azt határozza meg, hogy milyen gyakran férünk hozzá az adott területhez, megszorozva a lemezről való beolvasás költségével, végül elosztva a felhasznált bájtok számával. Talán észrevettél egy szándékos pontatlanságot az iménti meghatározásban: hogyan lehetnek a kisebb elemek forróbbak beolvasásuk teljesítményigényességének következtében? A gyorstár-hozzáférés hőmérsékletére egyéb meghatározások is elképzelhetők, jelen cikk esetében azonban ez tűnik a legmegfelelőbbnek.
Ha két különböző típusú dolgot tárolunk egy csomópontban, amelyeknek különbözik az átlaghőmérséklete, akkor két külön csomópontba helyezésükkel a gyorstárazás hatékonyabbá tehető. Tegyük fel, hogy R bájtnyi RAM áll rendelkezésre gyorstárazásra, míg D bájtnyi lemezterületünk van. Tegyük fel, hogy a kérések nyolcvan százaléka a legutoljára használt dolgokra hivatkozik, ezek a csomópont H (vagyis forró) bájtjai. A nagyobb teljesítmény érdekében a H-t az R-hez képest csökkentjük. Ha a gyakran használt adatokat egyenletesen szétszórod, akkor nagyobb méretű gyorstárra lesz szükséged, és a gyorstár is veszít a hatékonyságából.

Gyorstárazási alapelvek

Ha növeljük a csomópontok közötti hőmérséklet-eloszlást, akkor kisméretű gyorstárból is nagyobb teljesítmény tudunk kihozni. Ha kétfajta dolognak eltér a hőmérsékletátlaga, akkor külön csomópontokban helyezzük el őket, így növelve a rendszer egészében a hőmérséklet-eloszlást.
Ha minden egyéb egyenlő, és két különböző típusú dolgot tárol több példányban egyetlen csomópontban, akkor a gyorstár hatékonyabbá tehető, ha több különböző csomópontban osztjuk el őket.

Mutatók csomópontokra

A csomópontokra hivatkozó mutatók gyakran hívódnak meg az általuk elfoglalt bájtok számához hasonló gyakorisággal. Vegyük azt az esetet, amikor mindhárom átjáró esetében mutatókat kell használni, amelyek az alsóbb szinten lévő csomópontokat érik el, és kisebbek, mint azok a csomópontok, amikre hivatkoznak. Ha csak csomópontmutatókat helyezünk el, és a kulcsokat a belső csomópontokra korlátozzuk, a csomópontok sűrűsége nő. Mivel a mutatókra méretükhöz képest négyzetesen több alkalommal van szükség, mint a fájlok tartalmát tároló többi elemre, a mutatók és a cikkelyek adatai között jelentős a hőmérsékletátlaguk közötti különbség.
A korábban tárgyalt gyorstárazási alapelveknek megfelelően a kétfajta hőmérséklet-átlagú csoport (a mutatók és az elemek adatai) elágaztatása növeli a gyorstár hatékonyságát.
Most talán azt kérdeznéd, hogy miért nem a valódi hőmérsékletadatok alapján végezzük az elágaztatást a típus szerinti helyett? Azért, mert a típus csak a hőmérséklettel függ össze? Azt tesszük, amit a legegyszerűbb kódolni, nem csak a hőmérséklet-eloszlást véve alapul.

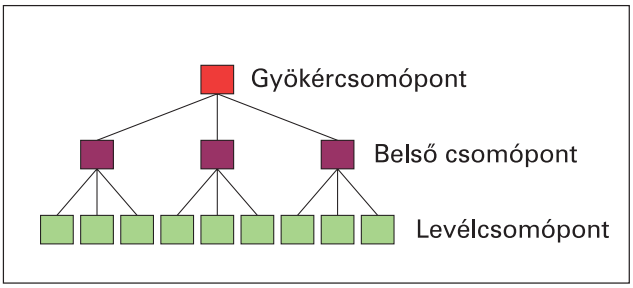
Néhány fatervezet úgy rendezi át a fát, hogy a magasabb hőmérsékletű elemek magasabban vannak, mint az alacsonyabb hőmérsékletű mutatók. Az elemadatok és a csomópontmutatók közötti hőmérséklet-különbség olyan mértékű, hogy az effajta tervezeteket én nem tartom versenyképesnek, emellett a megvalósításuk is bonyolultabb. A négyzetes nagyságú átlaghőmérséklet-különbséget véve alapul azt gyanítom, hogyha tévedek is, ez mégsem elegendő ahhoz, hogy foglalkozzunk vele. A többi fatervezet mintegy oldaljegyzetként megemlíti, hogy a vándorló elemek hőmérséklet szerint magasabban helyezkednek el a fában. Ha valaki pusztán csak a hőmérséklet szerint válogatna, de a szintek változtatása nélkül, az talán tényleg hatékonyabb lenne. Amennyiben valaki nem rendelkezne versenyképes tervezési alapokkal az elemek egymás körüli csoportosítására (ami néhány alkalmazás esetében igaz), és ha valaki az elemeket ahelyett, hogy egyenként férne hozzájuk, csomópontonként próbálja elérni, akkor érdemes lenne egy csomópont-újracsomagolót birtokolnia, hogy az elemadatokat hőmérséklet szerint csomópontokba rendezze.

A B+ fák jobban végzik a gyorstárazást, mint a B-fák

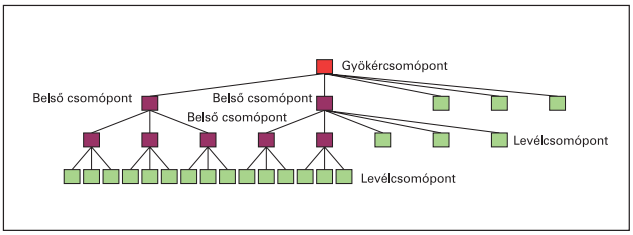
A B+ fák külön csomópontokban osztják el a mutatókat és az adatokat. A fában csomópontokra hivatkozó mutatók általában forróbbak, mint a fában található egyéb dolgok (mintegy négyzetes arányban). Mindazonáltal a korábban tárgyalt gyorstárazási alapoknak megfelelően a mutatók és az adatok szétválasztása hatékonyabb gyorstárazást eredményez. A számítógépiparban a B+ fákat a gyakorlatban jobbnak ismerik el, mint a B-fákat, mint ahogyan ezt a felvázolt elmélet is megjósolja. Az is elfogadott bölcsességnek minősül, hogy a kiegyensúlyozott fák hatékonyabbak a kiegyensúlyozatlan fáknál. Egyelőre azonban még nem elfogadott, de a fentiek alapján mégis megjósolható, a BLOB-ok – ahogyan az adatbázisipar hívja őket – rossz hatással vannak a teljesítményre. Erről és arról, hogy mik azok a BLOB-ok, hamarosan bővebben is szót ejtek.

Mit takar a kiegyensúlyozott?

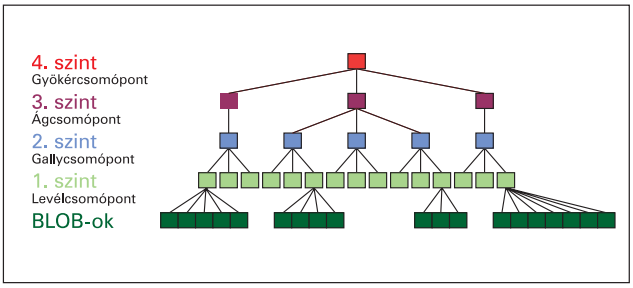
Ezt a fogalmat többféle, egymással kapcsolatban nem álló dologra is alkalmazzák, mint ahogyan a kiegyensúlyozott fák szakirodalma is megemlíti. E dolgok két legismertebbike a kiegyensúlyozott magasságok és a kiegyensúlyozott területek, amelyek a fa csomópontjaiban találhatók. Sajnálatos módon ezek a különféle meghatározások sok félreértést okoznak a szakirodalom olvasóinak – ezt én megpróbálom elkerülni.
A magasság-kiegyensúlyozott fák azok, amelyek esetében a gyökértől elindulva a levélcsomóponthoz vezető minden útvonal azonos hosszúságú. A hosszúság alatt a gyökércsomóponttól a levélcsomópontig bejárandó csomópontok számát értjük. Például az első ábrán a fa magassága 4, míg a *4. ábrán* a fa magassága 3, míg az egy csomópontból álló fa magassága 1. A legtöbb algoritmus a magasságmérés elvégzése céljából a fát csak a tetején növeli, ezáltal a fa sosem esik ki a magassági egyensúlyból.
Az *5. ábrán* egy kiegyensúlyozatlan fa látható. A fa kezdetben talán kiegyensúlyozott volt, de a különböző törlések következtében elvesztette néhány belső csomópontját. Illetve az is elképzelhető, hogy azért vált kiegyensúlyozatlanná, mivel anélkül lettek hozzá új elemek adva, hogy ügyeltek volna a kiegyensúlyozottságra.
A hagyományos adatbázismódszerekkel, ha egy csomópontnál nagyobb elemet szeretnénk tárolni, akkor ezt BLOB-okban tehetjük meg, ami azt eredményezi, hogy a fa kiegyensúlyozatlan lesz. A BLOB-ok jelentik az általános módszert arra, hogy



4. ábra Egy háromszintű fa

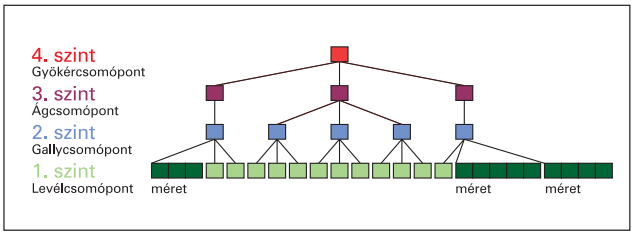


5. ábra Egy kiegyensúlyozatlan fa



6. ábra Négyszintű fa egy BLOB beszúrása után

nagyméretű elemeket tároljunk; működésük lényege pedig az, hogy a csomópont mutatókat tartalmaz azokra a csomópontokra, amelyek az elemet tartalmazzák. Az elemet tartalmazó csomópontokat levélcsomópontoknak hívják, ez minden "B*" fára jellemző.
A *6. ábrán* egy BLOB-ot szúrtunk be egy négyszintű fa levélcsomópontjába, ami annyit tesz, hogy a levélcsomópontban mutatókat helyeztünk el, amelyek a fájladatokat tartalmazó részekre hivatkoznak. A Reiser fájlrendszer 3-as változatának a fái így működtek.
Ez elég jelentős változás, amit mégis a teljes adatbázis-közösség elfogadott. Az itt leírt gyorstárazási alapelveknek megfelelően így csökkenthető a mutatók és elemadatok elágaztatása, de ez ront a gyorstár hatékonyságán. Emlékezzünk rá: a gyorstárazás alapelvei szerint ez rossz elgondolás. Tehát mindazon okokból kifolyólag, amelyek szerint a B+ fák jobbak a B-fáknál, a Reiser fájlrendszer 4-es változatának fái is jobbak a 3-as változat fáinál, bár nem olyan nagy mértékben.
Hogy ezt jobban megértsük, a *7. ábrán* egy Reiser4-es fa látható 3-as elágazási számmal, egy BLOB-bal az első szintű levélcsomópontban, valamint egy erre hivatkozó mutatóval a hármas szintű gallycsomópontban. Ebben az esetben a BLOB mezői folyamatosan helyezkednek el. A tárterület érdekében ez a többi levélcsomópont alatt helyezkedik el, de a hozzá tartozó mutató egy második szintű gallycsomópontban található, mint minden más elem mutatója is.
Bár elfogadott dolog, hogy a B+ fák jobbak a B-fáknál, az mégsem olyan elterjedt, hogy a BLOB-ok rosszul vannak megter-



7. ábra A Reiser4 a BLOB-okat az első szintű levélcsomópontokban tárolja

vezve, és az ezzel kapcsolatos tévhit jellemző az adatbázisiparra. Gray és Reuter azt mondja, hogy a külső memória keresésének az a jellegzetessége, hogy az egyes oldalak számát az általános (vagy leghosszabb) keresési úton csökkentsük, méghozzá olyan módon, hogy egy tetszőleges keresési útra csökkentjük az egyes oldalak számát, kisebbíti a valószínűségét annak, ha valamit közvetlenül a lemezről szükséges beolvasni (lásd a *Kapcsolódó címeket*).

Ezzel a magasság-kiegyensúlyozottság hatékonyságáról szóló elemzéssel az a bajom, hogy nem árulja el, hogy egy mérsékelten kiegyensúlyozott fával is lehet boldogulni – feltéve, hogy nem növeled meg nagymértékben a belső csomópontok számát. A gyakorlatban a kiegyensúlyozatlan fák csakugyan több belső csomóponttal rendelkeznek. A legtöbb mérsékelten kiegyensúlyozatlan fa esetében valamennyivel több memórián belüli lépdelésre van szükség a fában, ugyanakkor az I/O-terhelés nagymértékben nő, mivel több belső csomópontot kell feldolgozni, amiket – mivel így már túl sok csomópont van – nem lehet a gyorstárban tárolni.

De ha az összes BLOB-ot egyetlen helyen tárolnánk a fában, akkor ezáltal a csomópontok száma nem nőne jelentős mértékben, mivel az adatoknak minden más levélnél alacsonyabb szinten tartásából adódó teljesítménycsökkenés kis mértékű lenne. A fa bejárásának a költsége azonban – a RAM sebességétől függően – valamelyest nőne, de nem jelentős mértékben. A Reiser4-féle megközelítés nem az egyetlen lehetséges módja a belső csomópontszám csökkentésének. A BLOB-ok elágaztatása valószínűleg alapvetően megoldaná a gondot, ami azoknak a tervezőknek a hibájából keletkezik, akik a magasság-kiegyensúlyozott fák meghatározásának a lényegét figyelmen kívül hagyják. Valószínűleg az is nagy hatással lenne az I/O-teljesítményre, ha elkülönítenék azokat a BLOB-okat, amelyek nem állnak szoros kapcsolatban a fával. A Reiser4 esetében nem akartam arra korlátozni a fát, hogy az elemeket csak a méretük szerint ágaztassa el egymástól. De egy napon valaki majd biztos megpróbálja, és remélhetőleg elmondja nekünk, hogy mit tapasztalt.

A Reiser4 visszatér a magasság-kiegyensúlyozott fák klasszikus meghatározásához, vagyis minden egyes levélcsomóponthoz az út hossza egyenlő nagyságú. A Reiser4 nem színleli, hogy a nagyméretű elemeket tároló csomópontok nem a fa részei, annak ellenére sem, hogy a fa mutatókkal hivatkozik rájuk.

A Reiser4 a Reiser fájlrendszer 3-as változatához képest csökkenti a belső csomópontok és a mutatókat tartalmazó csomópontok számát. A 188 megabájtos Linux-rendszermag 2.4.1-es változatának forrásának tárolásához a ReiserFS v3 1629 csomópontot használt fel, ehhez a Reiser4-nek csak 164 csomópontra van szüksége. Ennek eredményeképpen a mutatók és csomópontok tárolásához szükséges RAM-mennyiség a Reiser4-nél döbbenetes mértékben mérséklődött.

Megjegyzés a cikk következő részeihez

A soron következő cikkekben megtudjuk, hogy miért lényegesen jobb a Reiser4 teljesítménye, mint a ReiserFS v3-é, még teljesen üres gyorstár mellett is. Megtudjuk, hogy miért jobbak a táncoló fák, mint a helyigényes kiegyensúlyozott fák, és azt is, hogy miként kezeljük a tranzakciókat olyan módon, hogy ugyanakkor nagymértékben csökkentjük a kétszer leírandó adatok mennyiségét.

Linux Journal 2003. május, 109. szám

Hans Reiser (reiser@namesys.com)
1979-ben csatlakozott az UC Berkeley-hez, és „Rendszerezésre" szakosodott, ami egy egyéni ágazat. Az elméleti modellek fejlődését tanulmányozza.

KAPCSOLÓDÓ CÍMEK

Jim Gray és Andreas Reuter Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers, Inc., 1993. – Ez egy régi, de jó könyv a tranzakciókról. Elérhető a ➲ http://www.mkp.com/books_catalog/catalog.asp?ISBN=1-55860-190-2 címen.