Taszkok ütemezése desktop-griden

FARKAS ZOLTÁN

MTA Számítástechnikai és Automatizálási Kutató Intézet
zfarkas@sztaki.hu

Lektorált

Kulcsszavak: taszk-ütemezés, skálázhatóság, hierarchikus desktop-gridek

A cikk keretein belül egy viszonylag új grid irányzattal, a desktop gridekkel kapcsolatos taszkütemezés kérdéseit mutatjuk be. A hagyományos gridekkel ellentétben desktop-gridek esetén nem egy adott infrastruktúrába küldi a felhasználó a taszkjait, hanem azok egy központi szerverre kerülnek, ahonnan az erőforrást felajánló donorokon futó kliensek letöltik, majd futtatja azokat. Tehát nem egy taszkhoz keresünk erőforrást, hanem a szabad kapacitással rendelkező erőforrás kér futtatandó taszkokat. A cikk során bemutatjuk a desktop-grideket, pár módszert azok skálázhatóságára, valamint bemutatjuk a hierarchikus desktop-gridekkel kapcsolatos ütemezési kérdéseket és lehetséges algoritmusokat.

1. Bevezetés

A hagyományos, szolgáltatásalapú gridek mellett egy másik grid irányzat mutat jelentős fejlődést: a desktop-gridek. A szolgáltatás alapú gridekkel [1] ellentétben a desktop-grid lényege az asztali számítógépek szabad számítási kapacitásának önkéntes felajánlásában és annak kihasználásában rejlik [2]. Vagyis egy desktop-gridbe bárki beléphet. Azonban a belépés szó értelme ebben az esetben más: míg a szolgáltatásalapú gridek esetén a belépő felhasználók használhatják a rendelkezésre álló infrastruktúrát, addig desktop-gridek esetén a felajánlott számítógépek alkotják az infrastruktúrát. A kihasznált szabad számítási kapacitásokért a felhasználók *krediteket* kapnak. A felajánlott számítógépek természetesen bármikor elhagyhatják a rendszert, ebből adódik a grid-jelleg. További különbség, hogy hagyományos gridek esetén a felhasználók tetszőleges alkalmazást futtathatnak a griden, desktop-gridek esetén a futtatható alkalmazások köre korlátozott: általában egy desktop grid egy probléma megoldására specializálódik, így egy alkalmazást futtat sok különböző paraméterrel.

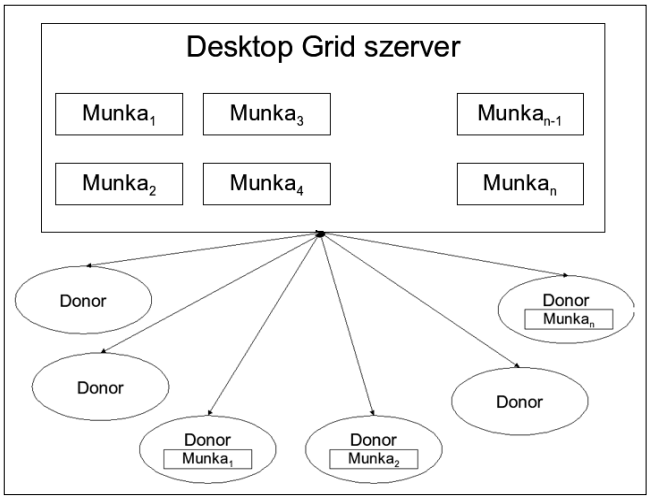
A desktop-gridek ideálisak olyan problémák megoldására, melyeknél egy nagyobb feladatot le tudunk bontani nagyszámú kisebb részfeladatra, ezek eredményéből pedig az eredeti probléma megoldása következtethető (master-worker típusú feladatok). Másik tipikus alkalmazási feladatosztály a parametrikus ellemzések (parameter study alkalmazások), ahol ugyanazt a feladatot kell nagyon sok, akár több tízezer paraméterrel lefuttatni. (Itt jegyezzük meg, hogy az ilyen parametrikus vizsgálatokat szolgáltatói grideken is el lehet végezni [11], vagy például a BME-n kidolgozott Saleve rendszerrel is végrehajthatók [12].

A master-worker típusú feladatok esetén, ha az eredeti probléma megoldása nagyon hoszszú ideig futna egy számítógépen (akár egy klaszteren, akár egy szuperszámítógépen), apróbb feladatokra leosztva viszont a részfeladatok számítási igénye annyira lecsökken, hogy viszonylag rövid idő (néhány óra, esetleg pár nap) alatt feldolgozható egy hagyományos PC-n. Desktop-gridekre példák a következő alkalmazások: közel 250 ország másfél millió számítógépének kapacitását használja a SETI@Home [3], mely a világűrből érkező rádiójelek feldolgozását végzi. Kisebb (bár korántsem elhanyagolható) volumenű projektek még az Einstein@Home [4] és a Climateprediction.net [5]. Hazánkban a nyilvános SZTAKI Desktop Grid (SZDG) futtat hasonlóan BOINC alapú nyilvános projektet, melynek célja a sokdimenziós bináris számrendszerek megtalálása [13]. Az algoritmust az ELTE Komputeralgebra tanszéke fejlesztette ki és az MTA SZTAKI-val közösen adaptálták az SZDG-re.

Az említett projektek közös vonása, hogy BOINC-ra [6] alapozva építették fel az infrastruktúrát. A BOINC a következő elven működik: egy központi szerveren található a projekthez kapcsolódó honlap, a futtatandó alkalmazás(ok) és az alkalmazás(ok)hoz kapcsolódó munkacsomagok. A munkacsomagok kaphatnak prioritást: nagyobb prioritási szint beállításával jelezheti a desktop-grid adminisztrátora, hogy számára az adott munka kiszámolása fontosabb, mint a többié. A felhasználók egy BOINC kliens telepítésével kapcsolódhatnak a szerverhez (így ajánlhatnak fel egy számítógépet – *donort*), ahonnan a kliens letölti a futtatandó alkalmazást, valamint adott mennyiségű munkacsomagot feldolgozásra. Amint van munkacsomag, a BOINC kliens elindítja az alkalmazást, amely feldolgozza a munkacsomagot. A feldolgozás végeztével a BOINC kliens feltölti a számolás eredményét a központi szerverre.

Az *1. ábra* mutatja egy desktop-grid felépítését.

A cikk további részeiben először röviden bemutatjuk az desktop-grid esetén felmerült taszkütemezéssel kapcsolatos kérdéseket és az azokkal foglalkozó cikkeket. Utána bemutatunk három módszert desktop-gridek számítási kapacitásának egyszerű növelésére, majd ezek közül egyet részletesen körüljárunk. Végül ejtünk pár szót a további lehetséges kutatási irányokról, majd röviden összefoglaljuk a leírtakat.



1. ábra A desktop grid felépítése

2. Ütemezési kérdések

A legfontosabb kérdések: mit ütemezzünk és miért, másként: mi az ütemezés célja? Az egyértelmű cél az, hogy a még fel nem dolgozott munkacsomagokat minél előbb kiszámolják a donor számítógépek. Vagyis a munkacsomagokat szeretnénk kiosztani olyan módon, hogy:

– a nagyobb prioritású munkacsomagok előbb kerüljenek feldolgozásra,
– a donorok lehetőleg minél kevesebbet dolgozzanak feleslegesen,
– a desktop-griden található összes munkacsomagot a lehető legrövidebb időn belül kell feldolgozni.

A bevezetés alapján feltehetjük még a kérdést, milyen ütemezéssel kapcsolatos kérdések merülhetnek fel desktop-gridek esetén? Több vonatkozásban beszélhetünk ütemezésről: egyrészt, kérdés, hogy mennyi munkát igényeljen egy donor (pontosabban a donoron futó BOINC kliens) [7,8].

Amikor egy donor munkát kap, a szerver megjegyzi, hogy kinek osztotta ki a feldolgozandó adatot és kiosztáskor egy határidőt rendel a munkacsomaghoz. A megadott határidőn belül vissza kell érkeznie az eredménynek a szerverre, különben a szerver azt feltételezi, hogy a donor valamiért nem tudta befejezni a feldolgozást és kiosztja más donornak a munkát. Tehát lényeges, hogy a donor csak annyi munkát kérjen, amennyit fel is tud dolgozni határidőre. Másrészt, egy donor több desktop-gridhez is kapcsolódhat, így kérdés, hogy az egyes desktop gridek között milyen arányban ossza meg szabad számítási kapacitását. Ezt az arányt a donort felajánló beállíthatja, vagyis ha úgy érzi, hogy két (vagy több) projekt közül számára az egyik valamiért különösen fontos, a szabad számítási kapacitás nagyobb részét ajánlhatja fel számára. Így a határidő betartását szorgalmazó ütemezés tovább bonyolódik: figyelembe kell venni az egyes projektek súlyát is.

Harmadrészt, felmerülhet a kérdés, hogy a szerver végez-e valamilyen ütemezést, azaz mely munkacsomagokat küldi ki először feldolgozásra? A jelenlegi BOINC implementáció elsősorban a munkacsomagok prioritása szerint csökkenő, másodsorban a munkacsomag létrehozási ideje szerint növekvő sorrend alapján osztja ki a munkacsomagokat feldolgozásra.

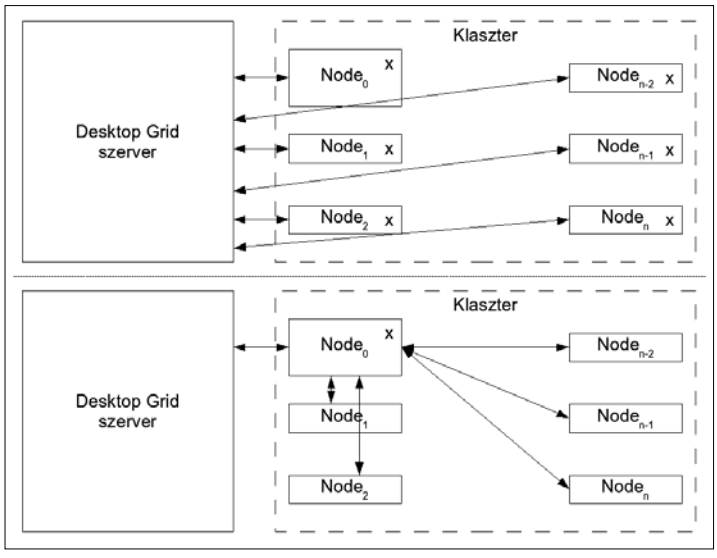
3. Desktop-gridek skálázhatósága

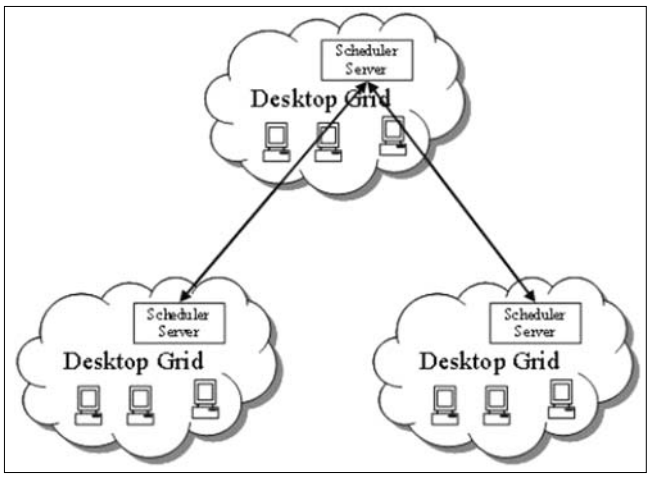
Felmerül a kérdés, meddig növelhető egy desktop-grid teljesítménye sima számítógépekkel, illetve mekkora teher egy-egy új számítógép felajánlása? Mint a bevezetőben láttuk, egy számítógép bekötése pár egyszerű lépést igényel csupán, de számítógépek százainak (pl. klaszter) bekötése már időigényes és monoton munka. A feladat egyszerűbbé tétele érdekében az MTA SZTAKI kifejlesztett egy speciális BOINC klienst, az úgynevezett klaszter klienst [9], mely hatalmas méretű klaszter felajánlása esetén is csupán egyetlen kliens telepítését igényli. A telepítés után a klaszterkliens feladata, hogy a klaszter számítógépeire szétossza a kapott feladatokat. A BOINC szerver szempontjából a klaszer egy többprocesszoros számítógépként látszik és ennek megfelelően is kér munkát a szervertől. A *2. ábrán* ‚X' jelöli a szükséges felajánlásokat klasztergépek egyenkénti, illetve klaszterklienssel végzett felajánlása esetén.

További bővítési lehetőség desktop-gridek számítási kapacitásának növelésére az, ha a desktop-grideket hierarchiába, fastruktúrába szervezzük. A struktúrában alsó szinten levő desktop-grid munkát igényelhet a felette levő desktop-gridtől. Így a hierarchia alkalmazásával lehetőség nyílik arra, hogy egy desktop-grid teljesítményét egy másik desktop-grid teljesítményével növeljük. A hierarchikus modell egy implementációját elkészítette az MTA SZTAKI [10].

A *3. ábra* példát mutat egy hierarchikus desktop-grid rendszerre, ahol a felsőbb szintű desktop-gridtől (például egyetem egy karának desktop-gridjétől) munkát kérnek az alsóbb szintű desktop-gridek (például az egyetemi kar tanszékeinek desktop-gridjei).

2. ábra Lehetséges klaszter-felajánlások





3. ábra Hierarchikus desktop-grid rendszer

Továbbgondolva a hierarchikus desktopgrid-modellt, a fastruktúra mellett lehetőség van egyenrangú desktop-gridek kialakítására, ahol az egyenrangú desktop-gridek tetszés szerint adhatnak át egymás között munkát. Ekkor a fenti ábrán látható alsó szintű desktop-gridek például munkát cserélhetnek egymás közt.

4. Skálázható desktop gridek ütemezési kérdései

Az előző részben három lehetséges módot mutattunk desktop gridek skálázására: klaszterek illesztése, hierarchia kialakítása, illetve egyenrangú desktop gridek összekapcsolása.

Klaszter illesztés esetén a klaszter kliens csupán anynyiban módosul az eredeti klienshez képest, hogy többprocesszoros számítógépként reprezentálja a klasztert. Ebből a szempontból klaszterek esetén a BOINC kliens ütemezési algoritmusa megfelelő, hiszen az fel van készítve több processzoros donorok kezelésére.

Az egyenrangú desktop gridek témaköre egyelőre koncepcionálisan létezik, így ezen rendszeren belüli taszkok ütemezésével a cikk keretein belül nem foglalkozunk.

Ebben a részben a hierarchiába szervezett desktop gridekkel kapcsolatos ütemezések kérdéseit tárgyaljuk részletesebben. Először bemutatjuk a hierarchikus kialakításból származó ütemezési problémákat, majd bemutatjuk azokat az eseményeket, amelyek egy hierarchikus rendszer állapotát (ez által az ütemezési algoritmusok működését) befolyásolják, végül bemutatunk pár ütemezési algoritmust, melyek hierarchikus desktop gridek esetén alkalmazhatóak.

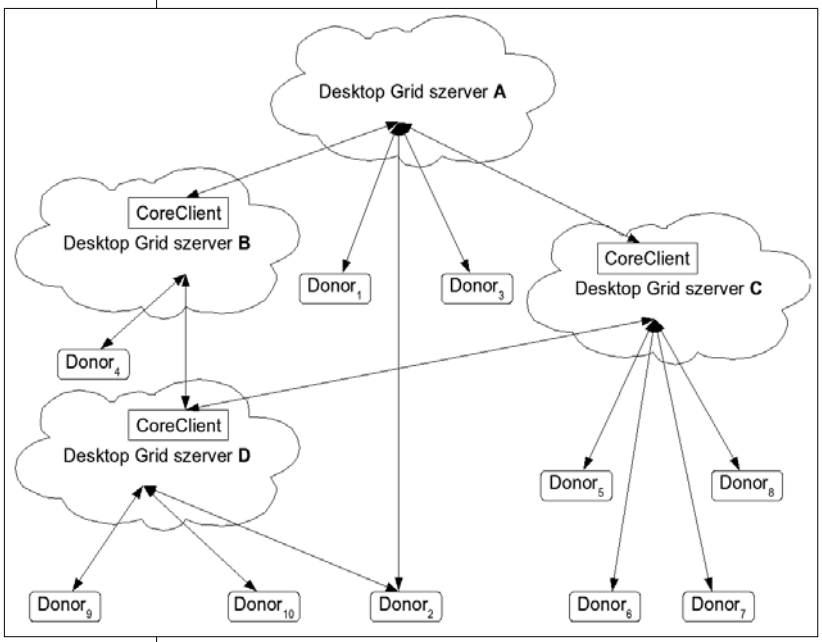
4.1. Hierarchikus desktop gridek ütemezési problémái

Hierarchikus desktop gridek esetén jelentkező ütemezési problémákat legegyszerűbben a *4. ábra* segítségével lehet bemutatni. Az ábra alapján a következő problémákat azonosíthatjuk: túl sok/kevés munka kérése felsőbb szintről, határidő túllépése és felsőbb szintekről származó munkacsomagok közötti különbségtétel.

Az első probléma akkor jelentkezik, amikor egy alsóbb szintű desktop grid által kért munka mennyisége nem tükrözi a desktop grid teljesítményét. Példaként a lenti ábra jelöléseit használva, ha a ‚B' desktop grid kis teljesítményű, de sok munkát kér az ‚A' desktop gridtől, akkor ezzel munkát vonhat el a nagyobb teljesítményű ‚C' desktop gridtől. Ennek ellenkező esete, amikor a sok donorral rendelkező desktop grid kevés munkát kér és a donorok nagy része „malmozik".

A második probléma akkor jelentkezhet, amikor egy desktop grid túlvállalja magát (mert túl sok munkát kér), és a letöltött munkacsomagok határideje lejár a felsőbb szintű desktop griden. Ezt az alsóbb szintű desktop grid egészen addig nem veszi észre, míg meg nem próbálja visszatölteni az eredményt felsőbb szintre. Vagyis, túl sok munka kérése esetén az alsóbb szint donorjai feleslegesen dolgozhatnak.

Hierarchikus desktop gridek esetén kérdés, hogy adjunk-e prioritást a felsőbb szintről érkező csomagoknak, illetve ha egy desktop grid több felső szinttől is kér munkát, melyiket részesítse előnyben, mely csomagjait dolgozza fel előbb? A felsőbb szintű desktop gridtől származó munkacsomagok prioritása könnyen biztosítható, ha az onnan származó munkacsomagok nagyobb prioritással kerülnek be az alsó szintű desktop grid szerverre, mint bármelyik, nem felső szintről származó munkacsomag prioritása.



4. ábra ösztetett hierarchikus rendszer

Hierarchikus rendszerben egy desktop grid több felsőbb szinthez is csatlakozhat. Ebben az esetben alkalmazható a BOINC kliens módszere: az alsóbb szintű desktop grid adminisztrátora megadhatja, hogy a munkacsomagok hány százaléka érkezzen az egyes felsőbb szintű desktop gridektől.

4.2. Hierarchikus desktop grideket befolyásoló események

Számos olyan esemény létezik, amely közvetlenül módosítja egy hierarchikus rendszer állapotát, például: donorok be- és kilépése, új munkacsomag megjelenése, munkacsomag átadása, munkacsomag feldolgozása, desktop grid be- és kilépése.

• Új donor megjelenése egy desktop grid teljesítményét növeli. A donor azonnal kapcsolódik a kérdéses szerverhez és onnan munkát kér, melyen elkezd dolgozni.

• Donor kilépése csökkenti a desktop grid teljesítményét. Sajnos ebben az esetben a desktop grid szerver nem értesül azonnal a kilépés tényéről, vagyis ha a donor kért korábban munkacsomagokat, azokat addig nem osztja ki a szerver újabb donoroknak, amíg azok határideje le nem jár.

• Új munkacsomag megjelenése többféleképpen hathat: ha a munkacsomag prioritása magas, akkor lehetőleg minél előbb meg kell kapnia egy donornak feldolgozásra. Ha nincs prioritása, akkor bekerül a várakozási sor végére.

• Munkacsomag átadás akkor következik be, amikor egy alsóbb szintű desktop grid felsőbb szintről kér munkát. Ekkor felsőbb szinten az átadott munkacsomagokhoz generálódik egy határidő, amin belül eredménynek kell érkeznie, különben feleslegesen dolgoztak az alsóbb szintű desktop grid donorjai.

• Desktop grid belépése többféle módon történhet: ha alsó szinten lép be egy desktop grid, akkor teljesítménynövelő szerepet tölt be. Felsőbb szintre történő belépéskor az alá bekapcsolt desktop grideknek plusz munkát jelent a tőle származó munkacsomagok feldolgozása, vagyis az ő szempontjukból saját csomag-feldolgozási teljesítményük csökken. Köztes szintre is beléphet az új desktop grid, ekkor szerepe kettős: fogad is és továbbít is munkacsomagokat.

• Desktop grid kilépése több dolgot eredményezhet. Ha felső szintű desktop grid lép ki, akkor munka tűnik el. Ekkor, ha alsóbb szint kapott munkát, feleslegesen számolja azt ki. Alsóbb szintű desktop grid kilépése azt eredményezi, hogy a felsőbb szintről kiosztott munkát addig nem kapja meg másik donor (vagy desktop grid), amíg határideje le nem jár.

4.3. Hierarchikus desktop gridek ütemezési algoritmusai

Ebben a részben röviden bemutatunk néhány lehetséges ütemezési algoritmust hierarchikus desktop gridek esetére, majd elemezzük, hogyan reagálnak a korábban bemutatott főbb állapotmódosító eseményekre.

A bemutatásra kerülő ütemezési algoritmusok közös tulajdonsága, hogy „lokális” algoritmusok, vagyis nincs rálátásuk a teljes hierarchikus rendszerre, hatókörük az egyes desktop gridekre korlátozódik, vagyis csak a hozzájuk kapcsolódó desktop gridről rendelkeznek információval, a felsőbb szintű desktop gridek állapotáról információjuk nincs.

4.3.1. Alapütemezés

Ez az „ütemezési” algoritmus a SZTAKI által létrehozott hierarchikus modell implementáció alapértelmezett algoritmusa. Lényege a következő: az alsó szintű desktop grid fix ‚n’ processzorral rendelkező donorként mutatja magát a felsőbb szintek felé, azaz a felsőbb szintekről származó, feldolgozás alatt levő munkacsomagok száma maximum ‚n’ lehet.

Az ütemezés több korábban említett problémát nem old meg: mivel előre kötött a kért munkák száma, ezért az algoritmus nem követi, ha a dekstop grid teljesítménye növekszik, vagy csökken. Így elképzelhető olyan extrém eset (például üres alsó szintű desktop grid esetén), amikor a frissen kapcsolódó donorok nem kapnak munkát. Ellenkező esetben (donorok kilépésekor), amikor a desktop grid teljesítménye csökken, túl sok munkát kér, így a felsőbb szintről származó munkák határideje rendre lejár: a donorok feleslegesen dolgoznak.

Amennyiben a desktop grid adminisztrátora követi a változásokat, módosíthatja a kért munka mennyiségét, de ez külső, emberi beavatkozást igényel.

4.3.2. Donorfüggő ütemezés

Ez az ütemezési algoritmus annyiban próbálja javítani az alap ütemezést, hogy a kért munkacsomagok száma követi az alsó szintű desktop gridhez kapcsolódó donorok számát. Vagyis, új donorok belépése vagy donorok kilépése esetén ez jó megoldás, az algoritmus alkalmazkodik a változáshoz. Donor kilépése esetén fontos, hogy a kilépő donort a felhasználója törölje a rendszerből, különben az algoritmus feltételezi, hogy a donor még mindig dolgozik az alsó szintű desktop grid számára. Vagyis donor kilépés esetén az algoritmus (hasonlóan az alap ütemezéshez) emberi beavatkozást igényel a helyes működéshez.

4.3.3. Aktívdonor ütemezése

A donorfüggő ütemezési algoritmus lehetséges kiegészítése egy olyan szűrés, amely csak az aktív donorok számára kér munkát, azaz az olyan donorok kiszűrése, melyek hosszabb idő óta nem jelentettek le kész munkát. Két lehetőség kínálkozik a passzív donorok szűrésére:

– a desktop grid adminisztrátor adott időközönként törli azokat a donorokat az adatbázisból, melyek az utolsó ellenőrzés óta nem töltöttek fel eredményeket, vagy
– az ütemezési algoritmus valósítja meg a donorok szűrését.

Az emberi beavatkozást elkerülendő célszerű az utóbbi megoldást választani. Kérdés, mikor kell az algoritmusnak egy donort paszszívnak tekintenie? A passzivitás meghatározása a következők alapján történik: az algoritmus passzívnak tekinti azokat a donorokat,

– amelyekhez nem tartozik munkacsomag, így kiszűrhetőek azok a donorok, akik nem kértek újabb munkát, továbbá
– azokat a donorokat, amelyekhez ugyan tartozik munkacsomag, de a munkacsomag feldolgozási határideje lejárt.

A fenti két feltétel vizsgálatával biztosan csak annyi donor számára fog munkát kérni az algoritmus, ahány a fentiek szerint aktív.

Az aktívdonor ütemezés tehát javítja a donorfüggő ütemezést: emberi beavatkozás nélkül képes kiszűrni a passzív donorokat, és az aktív donorok számától függő mennyiségű munkát kér.

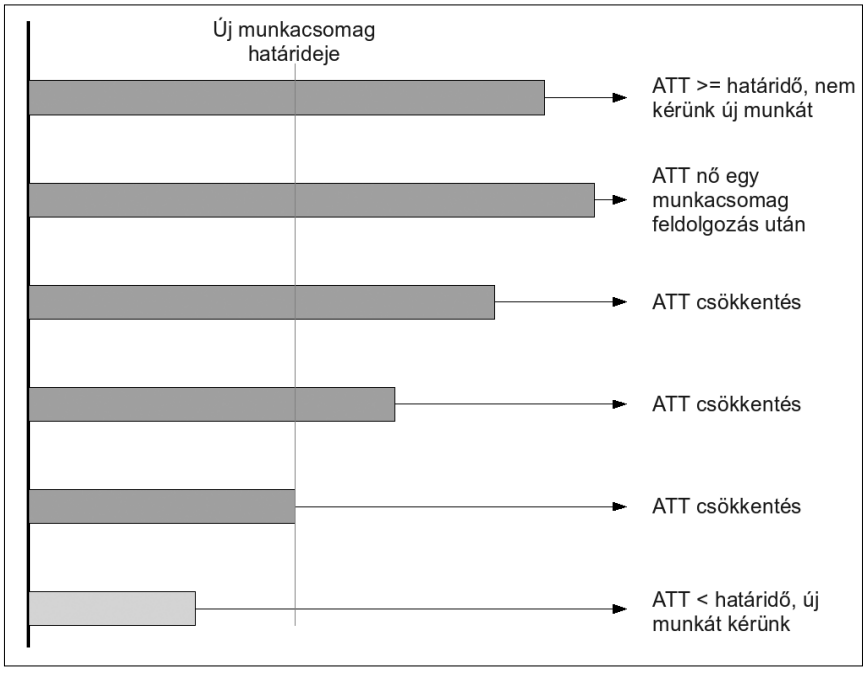
4.3.4. Timeout ütemezés

Az eddig említett alap, donorfüggő és aktívdonor ütemezési algoritmusok mindegyike a desktop grid állapotától, pontosabban a donorok számától függő mennyiségű munkát kért felsőbb szintről (az alap ütemezés bizonyos értelemben kakukktojás, hiszen fix számú munkacsomagot kér). Az említett algoritmusok hiányossága, hogy nem veszik figyelembe a felsőbb szintről kapott munkacsomagok határidejét, vagyis hiába kér annyi munkát, amennyi donor dolgozik, ha a donorok nem tudják teljesíteni a határidőket (esetleg azért, mert a donorok nagyon lassú számítógépek).

A probléma kiküszöbölése érdekében a Timeout algoritmus nyilvántartja a munkacsomagok átlagos megfordulási idejét (*Average Turnaround Time*, ATT): a munkacsomag megfordulási idők (*Turnaround Time*, TT – a munkacsomag felsőbb szintről való letöltése és a hozzá tartozó eredmény visszatöltése közötti idő) átlagát. A timeout ütemezés lényege, hogy az algoritmus folyamatosan frissíti az ATT értékét, azaz statisztikát készít a desktop grid teljesítményéből. Munkacsomag letöltésekor ellenőrzi a kapott munkacsomag határidejét. Amennyiben a kapott határidő kisebb, mint az aktuális ATT, eldobja a munkacsomagot és nem kér több munkát. Így azonban holtpontra juthat az algoritmus: ha nem kap újabb munkát, nincs ami az ATT-t csökkentse. Az ilyen helyzetek elkerülésére az algoritmus adott időközönként csökkenti az ATT értékét. Majd ha az ATT a legutolsó letöltött és eldobott munkacsomag határideje alá csökkent, ismét próbálkozik munka letöltésével.

Az *5. ábra* a timeout algoritmus működését mutatja.

5. ábra ATT csökkentése



4.3.5. Donortimeout algoritmus

A timeout algoritmus problémásan működhet abban az esetben, ha jelentősen eltérő teljesítményű donorok tartoznak a rendszerhez. Tegyük fel, hogy két donor dolgozik az alsóbb szintű desktop grid számára és felső szintről ugyanolyan számítási igényű és határidejű munkacsomagokat kap a desktop grid. Az egyik donor 10 perc alatt végez a munkával, a másik 50 perc alatt, és a munkacsomagok határideje 40 perc. Egyértelmű, hogy a lassú donor sosem fog érdemben végezni egy munkacsomaggal sem, az ATT kezdetben mégis 30 perc, vagyis feleslegesen kér az algoritmus munkát a lassú donor számára is.

A probléma kiküszöbölésére a donortimeout algoritmus donoronként tartja nyilván az ATT értékeket, ezáltal elkerülhető a fentebb említett gond: az első donor ATT értéke 10 perc, számára mindig kér újabb munkát az algoritmus, a lassú donor ATT-je viszont kezdettől fogva 50 perc, vagyis csak azon ritka alkalmakkor fog számára munkát kérni az algoritmus, ha a hozzá tartozó ATT értékét 30 perc alá csökkentette.

4.3.6. Várakozási sor

Az eddig bemutatott algoritmusok figyelték a donorok számát, esetleg figyelembe vették a munkacsomagok átalagos feldolgozási idejét, a kapott munkák határidejének viszonyát. Viszont nem foglalkoznak a desktop grid állapotával, azon belül a helyi, még feldolgozásra váró munkákkal. Abban az esetben, ha a felsőbb szintről kapott munka nem élvez prioritást a helyiekkel szemben, feldolgozásuk csak a helyi munka után történik meg.

A várakozási sor működése során figyelembe veszi az aktív donorok halmazát, a kapott munka prioritását, a kapott munka határidejét, valamint a feldolgozásra váró munkák halmazát. Új munka letöltésekor prioritás szerint sorba rendezi a létező munkacsomagokat és az aktív donorok ismeretében becslést ad az egyes munkák elvégzésének idejére (például pesszimista becslés esetén feltételezi, hogy mindig a leglassabb donorok kapják a munkát). Amennyiben a kapott munka nem dolgozható fel határidőn belül, elveti azt és adott ideig nem is kér munkát felsőbb szintről.

A várakozási sor algoritmus valójában több ütemezést is magába foglalhat: a létező munkák feldolgozási idejének becslési módszerétől függően más és más ütemezési algoritmust kapunk.

5. Összefoglalás

A cikk keretein belül a desktop gridekben található munkák ütemezésének kérdéseit és azok lehetséges megoldásait jártuk körbe. Először bemutattuk a desktop grid fogalmát és megemlítettünk pár olyan eredményt, mely

a munkák donor oldali ütemezésével foglalkozik, vagyis azzal a kérdéssel, hogyan végezhető el adott mennyiségű munka határidőn belül, a szabad számítási kapacitás kihasználásával.

Következő lépésként több módszert is ismertettünk a desktop gridek skálázására: klaszter bevonását, hierarchikus desktop grid építést és az egyenrangú desktop gridek felépítését. A három módszer közül a hierarchikus desktop gridekkel foglalkoztunk részletesebben: megvizsgáltuk, milyen problémákra kell odafigyelni az ütemezés során, bemutattuk mik befolyásolják az ütemezési algoritmusokat, végül példákat mutattunk több lehetséges ütemezési algoritmusra, melyek több szemszögből becsülik meg a letöltendő munka mennyiségét.

További munkák között szerepel az algoritmusok teljesítményének szimuláción keresztüli vizsgálata, valamint az egyenrangú desktop gridekkel kapcsolatos ütemezés körüljárása.

A fenti algoritmusokat az „Új generációs grid technológiák kifejlesztése és meteorológiai alkalmazása a környezetvédelemben és az épületenergetikában” című Jedlik Ányos projekt [14] keretében létrehozandó hierarchikus SZTAKI Desktop Grid rendszerekben fogjuk alkalmazni. A projekten belüli főbb alkalmazási területek: a klíma-modellezés és az épületek hűtéstechnikájának vezérlése. További fontos alkalmazása lesz a kidolgozandó ütemezésnek a CancerGrid EU FP6-os projektben, ahol rák elleni orvosságok fejlesztési idejének csökkentése a cél [15].

A hierarchikus SZTAKI desktop gridek megnyitják az utat a desktop gridek széleskörű és skálázható alkalmazásának irányába a kutatóhelyek és a vállalatok számára egyaránt.

Köszönetnyilvánítás

Jelen cikkben bemutatott munka az NKFP2-00007/2005 projekt keretein belül készült, amit a Nemzeti Kutatási és Technológiai Hivatal tett lehetővé a Jedlik Ányos program keretein belül az új generációs grid technológiák kifejlesztésére és meteorológiai alkalmazására a környezetvédelemben és az épületenergetikában.

Irodalom

[1] I. Foster, C. Kesselman, eds.,
The Grid: Blueprint for a New Computing Infrastructure.
Morgan Kaufmann, San Francisco, 1999.

[2] David P. Anderson:
Public Computing: Reconnecting People to Science,
Conference on Shared Knowledge and the Web.
Residencia de Estudiantes, Madrid, November 2003.

[3] D. P. Anderson, J. Cobb, E. Korpela,
M. Lebofsky, D. Werthimer:
SETI@home: An experiment in public-resource computing. Communications of the ACM,
November 2002, Vol. 45, No.11., pp.56–61.
http://setiathome.berkeley.edu/

[4] http://einstein.phys.uwm.edu/

[5] http://climateprediction.net

[6] David P. Anderson:
BOINC: A System for Public-Resource Computing and Storage.
5th IEEE/ACM Int. Workshop on Grid Computing,
8 November 2004, Pittsburgh, USA.

[7] Derrick Kondo, David P. Anderson, John McLeod VII:
Performance Evaluation of Scheduling Policies for Volunteer Computing.
3rd IEEE Int. Conf. on e-Science and Grid Computing,
Banagalore, India, 10-13 December 2007.

[8] David P. Anderson, John McLeod VII:
Local Scheduling for Volunteer Computing.
Workshop on Large-Scale, Volatile Desktop Grids (PCGrid 2007) held in conjunction with
the IEEE Int. Parallel & Distributed Processing Symposium (IPDPS), 30 March 2007, Long Beach.

[9] P. Kacsuk, N. Podhorszki, T. Kiss:
„Scalable Desktop Grid System”, High performance computing for computational science VECPAR’06,
Rio de Janeiro, Brazil, 10-13 July 2006, pp.1–13.

[10] P. Kacsuk, A. Marosi, J. Kovacs, Z. Balaton,
G. Gombas, G. Vida, A. Kornafeld:
SZTAKI Desktop Grid –
a Hierarchical Desktop Grid System,
Cracow Grid Workshop, Krakow, 2006.

[11] P. Kacsuk, G. Sipos, A. Tóth, Z. Farkas,
G. Kecskeméti, G. Hermann:
Defining and running Parametric Study Applications by the P-GRADE Portal.
Cracow Grid Workshop, Krakow, 2006.

[12] Zs. Molnár, I. Szeberényi:
Saleve: Simple Web-Services Based Environment for Parameter Study Applications,
6th IEEE/ACM Int. Workshop on Grid Computing,
Seattle, 2005.

[13] Burcsi Péter, Gombás Gábor, Kornafeld Ádám,
Dr. Kovács Attila, Kovács József, Marosi Attila Csaba,
Dr. Podhorszki Norbert, Vida Gábor:
„Szuperszámítógépes teljesítmény szuperszámítógép nélkül – A Binsys Projekt”,
Networkshop 2006.

[14] http://www.nkth.gov.hu/main.php?folderID=922

[15] http://www.cancergrid.eu/