ADATBÁNYÁSZ ALKALMAZÁS TÁMOGATÁSA A CLUSTERGRIDBEN

Vida Gábor, vida@sztaki.hu
Podhorszki Norbert, pnorbert@sztaki.hu
Kacsuk Péter, kacsuk@sztaki.hu
MTA SZTAKI, Párhozamos és Elosztott Rendszerek Laboratórium
H-1518 Budapest, Pf. 63

1 Tematika

A "Következő generációs adatbányászat nagy teljesítményű elosztott párhuzamos rendszereken" c. GVOP projekt fő célja egy nagy teljesítményű, elosztott, párhuzamos rendszereken működő adatbányászati szoftver prototípus kifejlesztése, lehetővé téve minden korábbinál jobb minőségű adatbányászati modellek létrehozását. A rendszer feladata, hogy a hálózati rendszer tulajdonságaihoz alkalmazkodva a rendelkezésére álló erőforrásokat a lehető legoptimálisabban használja fel, miközben az adatbányászattal foglalkozó szakembert mentesíti a párhuzamos feldolgozással kapcsolatos részletkérdések ismeretétől, miközben folyamatos tájékoztatást kap a részeredményektől, s ezek függvényében beavatkozhat a folyamatokba. A projekt keretében a nagy adat- és számításigényes adatbányász alkalmazásokat Grid rendszerek segítségével hajtjuk végre.

Az elosztott számítási környezet, konkrét Gridek elfedésére és a megfelelő funkciók biztosítására egy új programozói felületet definiáltunk a projektben (Distributed Computing API, vagy röviden DC API [1]). A mögöttes implementáció feladata az elosztott rendszer elérésével kapcsolatos feladatok ellátása, a működéshez szükséges hálózatkezelés, illetve az adatszolgáltatás megvalósítása. A rá épülő rétegek igényeit kiszolgálja, de elfedi az elosztott párhuzamos rendszer implementációjának részleteit. A DC API alapimplementációjának biztosítania kell azoknak az alapfunkciónak a működését, amelyek az első futtatható demo verzió futtatását lehetővé teszik.

A DC API első implementációját a ClusterGriden végeztük el és a jelen cikk célja ennek az implementációnak a bemutatása. A cikkben bemutatjuk a munkacsomagok létrehozásának és Gridben történő kiosztásának módját, a munkacsomagok végrehajtásának, felfüggesztésének, megszakításának és újraindításának technikáját. Az implementáció során a legnagyobb kihívást a részeredmények kezelése és ennek alapján a végrehajtás on-line vezérlésének megvalósítása okozta, ezért ennek megoldását részletesebben is taglalja a cikk.

2 Az adatbányászatról röviden

Az adatbányászat valójában egy gyűjtő-fogalom azon Mesterséges Intelligencia algoritmusokra, melyek képesek óriási adattömegből belső összefüggések automatikus feltárására.
A projekt célja egy nagy teljesítményű, elosztott, párhuzamos rendszereken működő

adatbányászati szoftver prototípus kifejlesztése. Az adatbányászati szoftverekre általában jellemző, hogy nagy futási idejűek, valamint, hogy a futás közbeni részeredményeik jó becslési lehetőséget biztosítanak arra, hogy megjósoljuk a futásuk végeredményét. Ezen okból kifolyólag egy olyan API rétegre (és alatta lévő Grid infrastruktúrára) van szükségünk, mely képes a kliens programok futás közbeni részeredményét visszajuttatni a központi (master) programhoz.
A ClusterGrid minden szempontból megfelelt a kívánalmaknak, vagyis:

1. Nagy számítási-kapacitással rendelkezik
2. Van mód a kliensek eredményeinek visszakérésére a rendszertől futás közben.

3 Distributed Computing API

A programozási interfész célja, hogy a párhuzamos programozáshoz nem értő fejlesztők is könnyedén tudjanak meglévő szekvenciális kódjukból egy többgépes rendszeren futó alkalmazást készíteni. A legegyszerűbb számítási modellt támogatja az interfész, azaz feltételezzük, hogy nincs szükség információcserére az egyes gépeken futó részprogramok között. A DC-API egy korábbi verziója már ki lett próbálva és le lett tesztelve a SZTAKI Desktop Grid-ben [2], mely egy a BOINC [3] infrastruktúrán alapuló SETI@home [4] jellegű Desktop Grid rendszer.

A jelenlegi feladat elvégzéséhez bővítettük a DC-API funkcióit és így lehetőség van a már futó részfeladatok felfüggesztésére, és azok futásának folytatására is. Erre a lehetőségre a korábbi verziónál nem volt szükség.

A DC-API feladatai:

- Alkalmazás inicializálása
  DC_init(projektnév, alkalmazásnév, konfigurációs fájl)
- Részfeladat létrehozása
  DC_createWU(kliensprogram, argumentum)
  DC_setInput(inputfájlok listája)
  DC_setPriority(prioritás)
- Részfeladat elküldése végrehajtásra
  DC_submitWU(részfeladat)
- Részfeladat megszakítása
  DC_cancelWU(részfeladat)
- Eredményre várakozás (blokkolt várakozás addig, amíg nincs eredmény vagy megadott ideig tartó várakozás)
  DC_checkForResult(timeout idő, eredményfeldolgozó eljárás)
- Részfeladat teljes törlése az alkalmazás memóriájából
  DC_destroyWU(részfeladat)
- Részfeladat felfüggesztése
  DC_suspendWU(részfeladat)
- Felfüggesztett részfeladat újraküldése végrehajtásra
  DC_resubmitWU(részfeladat)

Mint látható, ténylegesen csak néhány egyszerű feladatot kell elvégeznie az alkalmazásnak, csakis a részfeladatok megadására koncentrálva, és nem szükséges a végrehajtó rendszerrel törődnie. Az eredményekre várva egy függvényt kell definiálni,

amely egy részeredményt fel tud dolgozni. Ezt a függvényt a DC-API hívja meg, amikor egy részeredmény visszaérkezik az alkalmazás szintjére.

Az egyes architektúrák különbözőségeit a DC_init-ben megadható konfigurációs fájl tartalmazza, például az API-nak, a távoli rendszer eléréséhez szükséges parancsok elérési útvonala, az inputfájlok elérési útvonala, gépnevek, erőforrás menedzserek nevei, stb. A konfigurációs fájlok különbözőek lehetnek az egyes architektúrákra, de ez nem befolyásolja az alkalmazás működését, sem létrehozásának módját.

3.1 A workunit állapotai

Az előző fejezetben felsorolt függvények segítségével van lehetőség a DC-API-ban meghatározott WorkUnit egységfeladatok manipulációira. A létrehozástól egészen az eredmények feldolgozásáig. Ahhoz, hogy a fenti függvények működését együttesen megértsük és átlássuk, szükség van a WorkUnit azaz egységfeladat állapotainak egyfajta meghatározására. Ezek az állapotok egyrészt segítik a DC-API-t használó fejlesztőt abban, hogy a WorkUnit életciklusát a megfelelő módon megtervezhesse, másrészt segíti a DC-API csomag implementációjának elkészítésében.

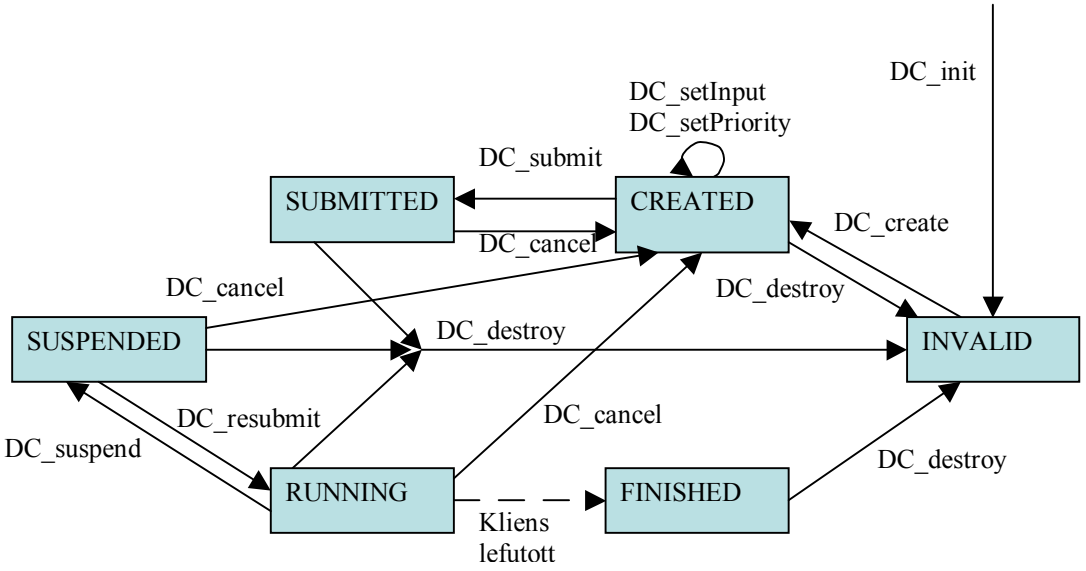
Egy WorkUnit a következő állapotokat veheti fel:

- INVALID
  Ez az állapot jelzi, amikor egy workunit-még nem lett létrehozva, vagy amikor már törölve lett. Logikailag az állapot értéke azt jelenti, hogy az adattábla e mezője üres és következő létrehozott workunit-ot itt lehet kialakítani.
- CREATED
  Ez az állapot minden esetben a WorkUnit életciklusának belépési pontja. Egy WorkUnit, akkor veszi fel ezt az állapotot, amikor megtörtént a WorkUnit-ot reprezentáló adatstruktúra létrehozása és minden minimálisan szükséges információval el lett látva ahhoz, hogy a DC-API azt kezelni tudja.
  A ClusterGrid-ben ekkor még nincs fizikailag létrehozva a job, csak logikailag létezik a DC-API memóriájában.
- SUBMITTED
  Ebben az állapotban lévő WorkUnit már átadásra került a futtatásért felelő Grid rendszernek, vagyis létre lett hozva egy JOBDIR struktúra és egy submit fájl. A WorkUnit ezen állapota az jelzi, hogy a részfeladatot megvalósító illetve reprezentáló kliens még várakozik a Grid rendszer végrehajtási várakozó sorában.
- RUNNING
  A WorkUnit ezen állapota azt jelzi, hogy a részfeladatot megvalósító, illetve reprezentáló kliens már allokálva lett a Grid egy megfelelő erőforrásán és azon megkezdte futását. Ez az állapot az alkalmazás számára láthatatlan. Csupán a DC-API tesz különbséget SUBMITTED és RUNNING állapot között.
- SUSPENDED
  A kliens (azaz WorkUnit) hosszabb, rövidebb ideig ideiglenesen felfüggesztésre került. Felfüggesztett állapotban a folyamat végrehajtása szünetel, de az addig elvégzett munka esetleg részeredmény nem vész el. Természetesen a felfüggesztés megfelelő működéséhez szükség van a kliens olyan irányú támogatására, hogy felfüggesztés esetén a saját számításához szükséges állapotterét képes legyen lementeni. Ezt felhasználói szintű

checkpointolással lehet megvalósítani, mely az alkalmazás fejlesztő feladata. Ekkor a ClusterGrid rendszerétől először visszakéri a DC-API a job eredményeit, majd eltávolítja a futó jobot a rendszerből.

- FINISHED
  A WorkUnit sikeresen lefutott, a számítási végeredmények rendelkezésre állnak.

A fentiekben ismertetett állapotok egymáshoz képesti viszonyát illetve azt, hogy az egyes DC-API WorkUnitokat manipuláló eljárások/függvények miként módosítják ezen állapotokat a következő ábra, szemlélteti. Az ábráról jól leolvasható továbbá, hogy az egyes állapotokban mely DC_* függvényhívások értelmezettek. Ha ezektől eltérő kombinációban kísérli meg az alkalmazás használni a DC_* hívásokat a viselkedés előre meg nem határozható, minden esetben implementációfüggő. A függvények jelen dokumentációban bemutatott implementációja minden ilyen esetben a hibás működést jelzi az alkalmazás számára, amit a DC_ERROR visszatérési értékkel jelez.



3.2 DC-Client API

A DC-Client API az egyes, számítási feladatokat végző Grid erőforrásokon a Grid keretrendszer és a kliensprogram közötti kapcsolatot teszi lehetővé. A legfontosabb feladata a felületnek, hogy az input és output fájlokat a kliensprogram megtalálja. A master alkalmazás minden egyes részfeladathoz létrehoz input fájlokat. Ezeket a fájlokat a Grid keretrendszer elszállítja a végrehajtás helyszínére, de ott a kliensprogramnak meg kell kérdeznie a keretrendszert, hogy az adott input fájl – amelynek nevét vagy parancssori argumentumként, vagy a kódba beégetve kapta meg – tényleges nevét és elérési útvonalát megismerje. Ugyanez vonatkozik az output fájlokra is a fordított irányban, a keretrendszert meg kell kérdezni, hogy hová és milyen néven tárolja el a kívánt nevű output fájlt. Csak így tudja a keretrendszer visszaszállítani az eredményeket a master-nek, az eredetileg definiált néven.

A DC-API feladatai:

- Fájlnév feloldás
  DC_ResolveFileName(type, logical_filename, physical_filename, maxlength)
  A kliens program megfelelő működéséhez elengedhetetlen a bemeneti adatok

fájlból történő olvasása és az elkészült számítási eredmények fájlba történő elmentése. Ahhoz, hogy ezeket a megfelelő helyen a megfelelő néven találja meg, segítségre van szüksége a futtató környezettől. Ezt a segítséget kapja meg ennek a függvénynek a segítségével. A függvény első paraméterében szereplő type változó értéke lehet 0, 1, 2 vagy 3 attól függően, hogy az input, output, ckpt, vagy temp fájl. A függvény ennek megfelelően képezi le, hogy a JOBDIR könyvtár rendszerben hol található a keresett fájl.

- Részeredmény visszaküldés
  DC_SendResult()
  A kliens program bármikor küldhet vissza részeredményeket a master-nek. Ez szükséges lehet, ha a beküldött kliens programok hosszú futási idejűek. Ekkor tájékoztatják a master-t, hogy hol tartanak a futásban, vagy hogy milyen eredményekkel tud szolgálni jelenleg.

- Checkpoint készítése
  DC_CheckpointMade()
  A kliens program ennek a függvénynek a segítségével tudja jelezni az infrastruktúra számára, hogy elkészített egy friss állapottér lementést (checkpoint). Ha azt az erőforrást, amin a kliens program fut, kikapcsolják a rendszerből, az addigi számítás eredménye elveszik. Ha a kliens időről időre a saját számításához szükséges állapotteret elmenti, a felfüggesztésből származó veszteség kiküszöbölhető. Ha a kliens végez állapottér mentést (felhasználói szintű checkpoint), akkor ezt a fenti függvénnyel jelezheti a rendszernek. A mentés lehetőséget ad a DC-API-nak arra, hogy a kliens újraindítása esetén, a kliens attól az állapottól folytassa futását, ahol a lementés történt.

- Kliensprogram kilépése
  DC_Finish()
  Egyes Grid keretrendszereknek szükségük lehet arra, hogy tájékoztassuk a program befejezéséről. A jelenlegi implementációban erre nem volt szükség, de a későbbiekre való elővigyázatosságból már most megtörtént a definiálása e függvénynek. Jelenleg nem ismert olyan paraméter, melyet valamilyen konkrét Grid keretrendszernek át kellene adni, ezért üres paraméterlistával definiáljuk a függvényt.

4 A ClusterGrid-ről röviden

A ClusterGrid infrastruktúra-szolgáltatás a ClusterGrid bróker-rendszeren keresztül vehető igénybe. Létre kell hozni minden majdani futtatandó jobnak egy job könyvtárat, illetve azon belül úgynevezett kötelező alkönyvtárakat.

A JOBDIR könyvtár, melyet a könyvtár-struktúra gyökerének nevezünk, kötelezően tartalmaz még egy „submit” nevű leíró-állományt is. Ez a submit fájl, mely a végrehajtás helyszínén leképződik a helyi ütemező (Condor) submit-fájljává, kötelezően a JOBDIR könyvtárban foglal helyet.

DC-API által használt ClusterGrid parancsok részletesebb leírása megtalálható annak felhasználói kézikönyvében [5].

5 DC-API a ClusterGriden

A DC-API ClusterGridre történő implementációja során az volt a cél, hogy a korábbi fejezetekben kifejtett funkcionalitásokat végrehajtsa a ClusterGrid interface használatával úgy, hogy ezt a speciális interface-t teljesen elrejti a felhasználó elől, hogy annak csak a szabvány függvényhívásokkal kelljen foglalkoznia elrejtve az infrastruktúra sajátosságaiból fakadó részleteket, mint például a könyvtárstruktúra létrehozását vagy a különböző fájlok generálását.

5.1 DC-API implementáció

A DC-API master oldali implementációja szétbontható 4 jól elkülöníthető részre:

1. Inicializálás
2. Részfeladatok létrehozása
3. Részfeladatok végrehajtása
4. Eredmények feldolgozása

A továbbiakban ezeknek a részeknek a külön-külön történő részletesebb bemutatása következik.

5.1.1 Inicializálás

A DC-API master felőli oldalának inicializálása a DC_Init függvényhívással történik meg, aminek hatására a DC-API kiolvassa és értelmezi a paraméterként kapott konfigurációs fájl tartalmát, valamint kezdő értékre állítja az adattábláinak mezőit.

5.1.2 Részfeladatok létrehozása

Ahhoz hogy részfeladatokat (workunit-okat) hozzunk létre, melyeket később végrehajtásra kívánunk beküldeni a Grid-be először is le kell foglalni egy mezőt a DC-API adattáblájából. Ezt a DC_CreateWU függvényhívással lehet megtenni, melynek visszatérési értéke a workunit mezőjének, az adattáblában lévő helyének a sorszáma. A továbbiakban ezzel a sorszámmal tudunk hivatkozni a kívánt workunitra (ez lesz a DC-API-n belüli azonosítója).
További paraméterként még meg kell adni a függvényhíváskor a futtatandó bináris fájl nevét, valamint azt, hogy milyen argumentum értékekkel hajtsa végre a Grid infrastruktúra.

A DC_SetInput parancs használatával lehetséges bemeneti fájlokat rendelni az egyes workunit-okhoz. Ekkor a DC-API feljegyzi magának, hogy melyik workunithoz, milyen bemeneti fájlt (vagy fájlokat) rendeltünk hozzá.

Definiálva van továbbá a DC_SetPriority függvény is, mely segítségével lehetséges prioritásokat rendelni az egyes workunitokhoz. Mivel ilyen funkció nem létezik a ClusterGrid bróker-rendszerében, így a DC_API ezen implementációjában ez a függvény nem lett megvalósítva.

5.1.3 Részfeladatok végrehajtása

Ha elkészült egy részfeladat (workunit) definiálása a DC-API interface hívásokon keresztül, a DC_submitWU függvényhívás hatására a DC-API létrehoz neki egy JOBDIR könyvtár-struktúrát a ClusterGrid-nek megfelelő előírások alapján, valamint a hozzá tartozó submit fájlt. A futtatandó állományokat és azok bemeneti fájljait

átmásolja a megfelelő alkönyvtárakba, majd a clgr_submit parancs használatával beküldi a Grid-be végrehajtásra, végül a clgr_submit kimentén kapott grid azonosítót elmenti magának.

Ha egy már futó workunit-ra nincs szükség a továbbiakban, akkor a DC_cancelWU parancs kiadásával a DC-API leállítja annak futását (clgr_rm), valamint törli a létrehozott JOBDIR könyvtárrendszert is. A DC_submitWU újbóli kiadásával a workunit-ot újra be lehet küldeni a Gridbe végrehajtásra.

Lehetőség van a futások felfüggesztésére is. A DC_suspendWU hasonlóan a DC_cancelWU parancshoz leállítja a workunit futását a Gridben (clgr_rm), de előtte visszakéri a Grid-től annak aktuális eredményeit (clgr_getout). Amennyiben a részfeladatot úgy írták meg, hogy időről-időre készítsen felhasználói szintű checkpointot, úgy ebben az esetben egy újbóli beküldés (DC_resubmitWU) esetén az addig elvégzett munka nem vész el.

Amennyiben egy workunit-ra a továbbiakban egyáltalán nincs szükség, a DC_destroyWU használatával lehetséges leállítani annak esetleges futását (clgr_rm), valamint törölni a JOBDIR könyvtárát ugyan úgy, mint a DC_cancelWU használatával, ám ebben az esetben törlődik a DC-API adattáblájából is, minden beállításával együtt. Hosszan futó és sok részfeladatot készítő alkalmazások esetében feltétlenül szükséges a használata, hogy a DC-API adattáblájában (memóriájában) ne tárolódjanak, fölösleges adtok.

5.1.4 Eredmények feldolgozása

A Gridbe végrehajtásra beküldött workunit-ok eredményeit a DC-API-tól a DC_checkForResult függvény meghívásával lehet lekérni. Ennek hatására a DC-API a konfigurációs fájlban meghatározott érték szerinti időközönként lekérdezi a rendszertől, hogy a már beküldött jobok lefutottak-e. Ha talál lefutott jobot, akkor annak eredményét visszakéri a rendszertől (clgr_getout), majd meghívja a master alkalmazás callback függvényét, melyet a DC_checkForResult függvény paraméterében kellett átadnia a DC-API-nak.
A DC_checkForResult futása lehet blokkolt, vagy egy meghatározott ideig tartó, a ’timeout’ paraméter értékétől függően. (0 esetén blokkolt, nem nulla esetén annyi másodpercig tartó ciklikus lekérdezés.)

5.2 Implementációs nehézségek

A tematikában említett projekt keretein belül olyan alkalmazások futtatása a cél, melyik hosszú futási idejük alatt folyamatosan részeredményeket generálnak, valamint olyan master alkalmazás írása, mely a beküldött WorkUnit-ok folyamatosan visszaérkező részeredményeiből hoz döntést, a további futásra vonatkozóan. Ennek megvalósításához arra van szükség, hogy a job-ként végrehajtásra beküldött WorkUnit-ok jelezhessék részeredményeiket a Grid infrastruktúrán keresztül a master irányába.

Ahhoz, hogy egy job a DC-API (és ez által a master) tudomására hozhassa, hogy elkészült egy részeredménnyel, amit szeretne úgy visszajuttatni, hogy a futását közben nem szakítja meg, szükség van rá, hogy hirdetéseket tudjon készíteni, vagyis valamilyen plusz információt tudjon hozzájuttatni a státusz listájához. ClusterGriden ez a Mercury Grid monitorozó rendszer[6] segítségével történik.

Amennyiben, a kliens alkalmazás jelzi a DC-Client API-nak, hogy részeredményt szeretne visszaküldeni a master számára, a DC-Client API egy előre meghatározott adatfájlt hoz létre, melyet a Mercury rendszer észrevesz, és tartalmát hozzáadja a job státusz-listájához. Ezt a plusz információt keresve a (master oldali) DC-API észreveszi a változást, és a clgr_getout parancs segítségével futás közben visszakéri a job munkakönyvtárát a Grid rendszertől. Ez a lekérdezés nem befolyásolja a job-ot a további futásban, így az, változatlanul folytathatja a munkáját, miközben a master alkalmazás megkapja a job futás közben lemásolt könyvtárszerkezetét és az abban található részeredményeket.

Ezt a fajta hirdetés készítési módot, hogy egy job a Mercury rendszer segítségével tudjon plusz információt juttatni a státusz listájához kifejezetten a DC-API ClusterGriden történő implementációja miatt készítették el a ClusterGrid teszt-rendszerében ahhoz, hogy a GridML projektben szükséges részeredmény visszahozatalt támogatni tudjuk.

6 Összefoglaló

A ClusterGridre sikeresen implementált DC-API lehetővé teszi a projekt céljaként kitűzött adatbányászati szoftverek elkészítését, valamint más alkalmazásoknak is segítséget nyújthat a ClusterGrid használatában.
Ezen kívül az API megírásakor arra külön figyelmet fordítottunk arra, hogy elkülönítsük a ClusterGrid specifikus kódrészleteket, így megteremtve annak lehetőségét, hogy más Grid infrastuktúrákra is könnyen implementálhassuk ezt a programozói felületet.

7 Hivatkozások

[1] Podhorszki Norbert, Vida Gábor:
Alkalmazói programozási felület SETI-jellegű elosztott programokhoz és végrehajtó rendszer a BOINC infrastruktúrára
Networkshop 2005
http://nws.iif.hu/ncd2005/index.htm

[2] Sztaki Desktop Grid: http://szdg.lpds.sztaki.hu/szdg

[3] BOINC: http://boinc.berkeley.edu

[4] SETI@home: http://setiweb.ssl.berkeley.edu

[5] Stefán Péter, Szalai Ferenc Vitéz Gábor:
A ClusterGrid Infrastruktúra használata: Felhasználói kézikönyv
http://www.clustergrid.hu/docs/pdf/user_help.pdf

[6] Mercury Grid Monitoring System: http://www.lpds.sztaki.hu/mercury/