Eötvös Loránd Tudományegyetem
Informatikai Kar

Programozási Nyelvek és
Fordítóprogramok Tanszék

Adatintenzív alkalmazások optimalizálása GRID rendszerekben

Készítette:
Takács Zoltán

Témavezető:
Horváth Zoltán

Budapest, 2006. június 1.

Témabejelentő

Tartalomjegyzék

Köszönetnyilvánítás **5**

1. Bevezetés **6**

2. Áttekintés **8**
2.1. A GRID rendszerek fejlődése 8
2.1.1. Szuperszámítógépek 8
2.1.2. Klaszterek 8
2.1.3. Desktop GRID 9
2.1.4. Production GRID 9
2.1.5. Globus Toolkit 2 (GT2) alapú GRID-ek 12
2.1.6. Szolgáltatás-alapú GRID 13
2.1.7. Globus Toolkit 4 (GT4) 14
2.1.8. gLite 16
2.1.9. Multi-ágens alapú GRID 16
2.2. GRID alkalmazások 17
2.2.1. Számításigényes alkalmazások 17
2.2.2. Adatintenzív alkalmazások 17
2.2.3. Szekvenciális alkalmazások 17
2.2.4. Párhuzamos alkalmazások 18
2.2.5. Workflow alkalmazások 18
2.3. Fájlelérés a GRID rendszerekben 19
2.3.1. LCG-2 GRID 19
2.3.2. gLite 20
2.4. Optimalizálás 21
2.4.1. Adatok másolása és a job életciklusa 21
2.4.2. Workflow alkalmazások optimalizálása 21

3. Elemzések **23**
3.1. Szolgáltatás és ágens alapú megközelítések összehasonlítása . . 23
3.1.1. A szolgáltatás 23

3.1.2. Az ágens 23
3.1.3. Szolgáltatások rendszerének felépítése 24
3.1.4. Ágensek rendszerének felépítése 25
3.1.5. Szolgáltatás-alapú GRID felépítése 26
3.1.6. Ágens-alapú GRID felépítése 26
3.1.7. Összehasonlítás 27
3.2. A lokális háttértár használatának elemzése 28
3.3. Dinamikus fájleléréssel történő optimalizálás elemzése 29
3.3.1. Megoldás szálak segítségével 29
3.3.2. Új a cache-elést végző komponens készítése 30
3.3.3. Megoldás ágensek segítségével 31
3.3.4. Összehasonlítás 31
3.4. Ágensekkel történő optimalizálás elemzése 31
3.4.1. Megoldás egy ágens használatával 32
3.4.2. Megoldás két ágens használatával 32
3.4.3. Elemzés és összehasonlítás 32
3.5. A workflow esetében felmerülő lehetőségek elemzése 33
3.5.1. Ideiglenes fájlok kezelésének optimlizálása 33
3.5.2. Optimalizálás a rákövetkezők indítási idejének hangolásával 36
3.6. Az elemzés eredményeinek összefoglalása 37

4. Megvalósítási javaslatok 39
4.1. A dinamikus elérést biztosító komponens 39
4.2. Ágensek megvalósítása 40
4.3. A JADE rendszer 41
4.3.1. JADE rendszer tesztelése 42
4.3.2. JADE ágensek a jelenlegi GRID rendszerekben 45
4.4. AgentScape 45
4.4.1. A rendszer felépítése 45
4.5. Ágensek megvalósítása szolgáltatások segítségével 48
4.6. Ágensek megvalósítási lehetőségeinek összehasonlítása 49
4.7. A tűzfalak szerepe és problémái 50
4.7.1. Inteligens tűzfal 50
4.7.2. Proxy komponens 51
4.7.3. A lehetőségek összehasonlítása 52

5. Összegzés 54

Ábrák jegyzéke

2.1. A Desktop GRID szerkezeti felépítése 10
2.2. A Production GRID-ek elvi felépítése. 12
2.3. Szolgáltatások igénybevétele. 13
2.4. A fájl- és replika-katalógus felépítése az LCG-2 GRID-ben. . . 20

3.1. A hálózati és a lokális fájlelérés összehasonlítása 28
3.2. A virtuális SE elhelyezkedése a rendszerben. 35
3.3. A statikus és a dinamikus fájlelérés összehasonlítása. 37
3.4. Az optimalizálásban résztvevő komponensek elhelyezkedése. . . 38

4.1. Az AgentScape rendszer koncepciója 46
4.2. Az AgentScape architektúrája. 47
4.3. Az inteligens tűzfal működése. 51
4.4. A proxy komponens működése. 51

5.1. OptorSim szimuláció eredményeképpen kapott átlagos futási idők. 56

Köszönetnyilvánítás

Szeretnék köszönetet mondani mindazoknak, akik valamilyen formában segítettek abban, hogy elkészítsem a diplomamunkámat. Külön köszönöm azon oktatók áldozatos munkáját, akik megalapozták a szükséges tudásomat ahhoz, hogy képes legyek ezen munka elkészítésére. Legnagyobb köszönettel Horváth Zoltánnak tartozom azért, hogy elvállalta a témavezetést, segítséget nyújtott a szükséges irodalmak megtalálásában és tanácsokkal látott el a diplomamunka megírásával kapcsolatban.

1. fejezet

Bevezetés

A tudomány számos területén találkozunk olyan problémákkal, melyek megoldásához nagy számítási-, vagy tárkapacitásra van szükség, esetleg egy különleges eszközt szeretnénk használni egy távoli helyről. Ezeknek a problémáknak a megoldásához nagy segítséget nyújtanak a GRID rendszerek.

Azon jobok esetében, melyek nagy mennyiségű adattal dolgoznak, nagyon fontos az adatok elérésének optimalizálása. Ezen munkában az adatok elérésének optimalizálásának lehetőségeit vizsgálom.

Az adatintenzív alkalmazások esetében számos stratégiát alkalmazhatunk. Optimalizálhatunk futási időre, adatforgalomra, vagy akár a minél korábbi befejezésre. A legtöbb GRID rendszerben jelenleg használt stratégiák az erőforrások minél hatékonyabb kihasználását tartják szem előtt, de nem fordítanak kellő hangsúlyt arra, hogy a felhasználó minél gyorsabban megkapja a számítások eredményét.

A optimalizálási lehetőségek vizsgálatánál az volt a legfőbb szempont, hogy a GRID rendszerbe olyan komponenseket vezessek be, ami egy logikus felépítést eredményez, és a különböző stratégiák könnyű kivitelezését lehetővé teszi. Tehát egy általános felépítést javaslok a fájlelérésben résztvevő komponensek esetében, melyek lehetővé teszik a fájlok dinamikus elérését, előreolvasását és a workflow alkalmazások esetében az ideiglenes fájlok létrehozás közbeni olvasását. Ezen lehetőségek használatával optimális esetben elérhetjük, hogy a job végrehajtása közben a szükséges adatokat minden esetben a végrehajtást végző gép lokális háttértárolójáról olvashassuk várakozás és a kommunikáció költsége nélkül.

A 2. fejezetben egy áttekintést szeretnék adni a GRID rendszerek fejlődéséről és típusairól. Ezt követi a GRID alkalmazások típusainak áttekintése, majd a GRID-ekben használt fájlkezelést tekintem át, ami az adatintenzív alkalmazások szempontjából döntő fontossággal bír. Végül néhány optimalizálási javaslatot mutatok be.

A 3. fejezet fő célja a lehetőségek elemzése. Az első részében a szolgáltatások és az ágensek segítségével felépített rendszerek összehasonlítását végzem el, majd utána az optimalizálási javaslatok lehetséges megoldásaival foglalkozok.

A 4. fejezetben néhány megvalósítási javaslatot találhatunk, valamint ezzel együtt megvizsgálok néhány lehetőséget az ágens-alapú GRID-ek felépítésére. E célból bemutatok néhány már elkészített keretrendszert, melyek az ágensek megvalósítására nyújtanak megoldást.

Az 5. fejezet tartalmazza a munkám összegzését.

2. fejezet

Áttekintés

2.1. A GRID rendszerek fejlődése

A tudomány számos területén találkozhatunk olyan problémákkal, melyek megoldásához óriási mennyiségű számítási kapacitásra vagy tárhelyre van szükség. Egyetlen számítógép kapacitása nem elegendő ezen feladatok megoldására. Ezek a problémák inspirálják a fejlesztőket, hogy olyan rendszereket hozzanak létre, mely egyre nagyobb számítási és tárkapacitást képesek rendelkezésre bocsátani biztonságos és hibatűrő módon.

2.1.1. Szuperszámítógépek

Az első jelentős előrelépést a szuperszámítógépek jelentették, melyek nagy számú processzort, memóriát és háttértárat tartalmaznak. A szuperszámítógép esetében egy homogén rendszerről beszélhetünk. Ezen megoldásnak fizikai határai vannak és nagyon sokba kerülnek.

2.1.2. Klaszterek

A GRID rendszerek felé vezető úton az első lépés a klaszterek megjelenése volt, melyek személyi számítógépekből álló lokális hálózattal összekötött rendszerek.

Ebből láthatjuk, hogy a memória elosztott, és a kommunikáció nem osztott változók, hanem hálózati kommunikáció segítségével történik.

A legjelentősebb szoftvercsomag a Condor [2], mely egy klaszter erőforrásainak kihasználását, és a jobok optimalizált futtatását tette lehetővé.

Lehetőség van egy job több különböző paraméterrel való végrehajtásárra, párhuzamos alkalmazások futtatására, vagy akár workflow alkalmazások végrehajtására is.

A Condor egy adott hálózaton belül a ki nem használt processzorkapacitás hasznosítását teszi lehetővé. A rendszer a felhasználók távollétében jobokat futtat a tétlen gépeken.

Ehhez a szoftvercsomaghoz több kiterjesztés is készült, mely lehetővé teszi a Condor által vezérelt klaszterek összekötését, vagy akár GRID erőforrások használatát is.

2.1.3. Desktop GRID

A Desktop GRID a Condor filozófiájának bizonyos szempontból való általánosítása. A Desktop GRID esetében az erőforrást felajánlók(donorok) száma sokkal nagyobb, mint az ezetek felhasználók száma. Ilyen rendszer például a SETI@home [3], melyben a számítógép tulajdonosok milliói ajánlják fel számítógépeik kihasználatlan kapacitását egy probléma megoldása érdekében.

Itt láthatjuk, hogy az internet segítségével történik a kommunikáció, és a donorok a világ minden részéről ajánlanak fel erőforrást.

Nincs szükség a donorok gépeinek adminisztrálására. A felajánlónak csak fel kell telepíteni egy adott szoftvercsomagot, és a rendszer máris működőképes.

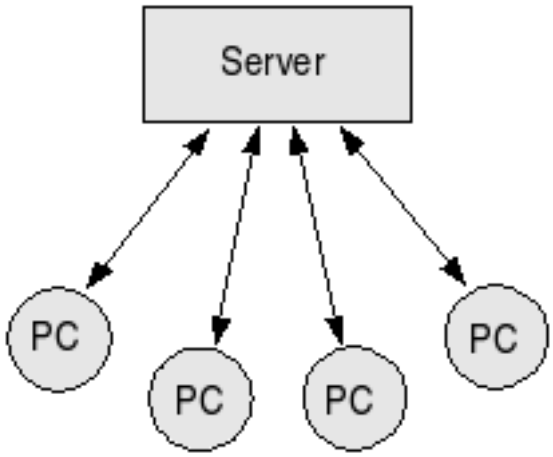
Ezen rendszerek két fő részből állnak (2.1 ábra). Az egyik egy központ, mely biztosítja a szükséges feladatokat, és az eredmények feltöltésének lehetőségét, valamint a felajánlók személyi számítógépei, melyeken a feltelepített szoftver észleli, hogy a gép tétlen, és a központtól kér egy feladatot. Ha a felajánlónak szüksége van a gépére, akkor a számítások felfüggesztésre kerülnek. Ha az eredmény kiszámítása megtörtént, akkor az eredményt feltölti a rendszer a központhoz.

Ezen rendszerek üzemeltetésénél fontos, hogy a felajánlók lelkesedését meg tudjuk tartani. Ennek érdekében ranglistákat és pontosztást alkalmaznak, mely arra ösztönzi az embereket, hogy minél többet számoljon a felajánlott számítógépük. Ennek fokozása érdekében csoportokat is képezhetnek az emberek, aminek eredménye képpen több gépet is felajánlanak, vagy ismerőseiket is bevonják a rendszer használatába.

Manapság nagy számú Desktop GRID működik a világon, melyek, valamilyen speciális probléma megoldásán dolgoznak.

2.1.4. Production GRID

A Production GRID a desktop GRID-del ellentétes filozófiát követ. A donorok száma kicsi. Általában intézmények ajánlják fel és tartják karban gépeiket.



2.1. ábra. A Desktop GRID szerkezeti felépítése

A Desktop GRID-del ellentétben itt nagy feladat a gépek adminisztrálása. A felajánló intézményeknek rendszergazdákat kell alkalmazni, hogy biztosítsák a folyamatos működést.

A felhasználók száma nagy lehet, mivel szinte bárki használhatja a GRID által nyújtott lehetőségeket.

A felhasználók Virtual Organization-ökben [4] (VO) csoportosúlnak. A VO a közös érdekeltségű felhasználók és a hozzájuk tartozó erőforrások adminisztrációját végzi.

Ha egy felhasználó használni szeretné a GRID-et, akkor egy GRID tanusítványt kell igényelnie. Ennek segítségével van lehetősége a GRID erőforrásainak használatára.

Jelenleg a GRID-ben tetszőleges alkalmazásokat futtathatunk a rendelkezésünkre álló gépeken. Ez biztonsági szempontból nagy veszélyeket hordoz magában. Ezen veszélyek csökkentése érdekében, csak indokolt esetben és megbízható személyek részére adnak ki tanusítványokat.

Ebben a rendszerben klasztereket, szuperszámítógépeket vagy akár speciális eszközöket is összeköthetünk az internet segítségével. Ez által tetszőleges eszközt egy adott GRID részévé tehetünk, így az erőforrások száma nagyságrendekkel növelhető a klaszerekhez képest, valamint kilépünk a lokális hálózatok által összekötött esztözök világából a világ méretű rendszerek világába.

A rendszer komponensei több típusúak lehetnek a feladatuknak és az absztrakciós szintnek megfelelően.

Worker Node(WN)

A GRID azon komponenseit nevezzük WN-nak [1], melyek a valódi számításokat végzik. Általában egy klaszter részei.

Storage Element(SE)

A SE [1] egy a megbízható és tartós adattárolásra használható komponense a GRID-nek. A fizikai tárolást végző tároló eszközök egy közös absztrakciója. Egységes elérést biztosít például a szalagos egységekhez vagy a merevlemezekhez.

Computing Element(CE)

A CE [1] egy absztrakciós szint, mely általában az egy klaszterben található WN-ok ütemezését végzi. Egy végrehajtási sor, melyben elhelyezhetjk a futtatni kívánt jobokat, majd a CE a megfelelő módon ütemezi őket a hozzárendelt gépek felhasználásával.

Resource Broker(RB)

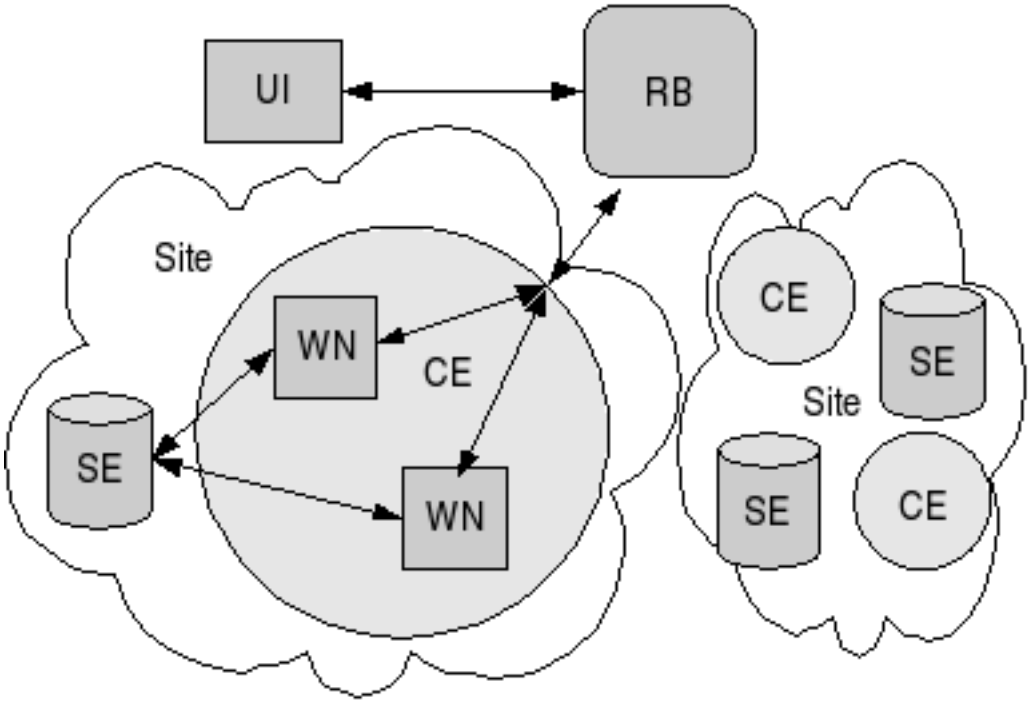
A RB [1] a rendszer globális ütemezője. Eldönti, hogy az adott job, mely CE-n kerüljön végrehajtásra. Értelmezi a jobokat leíró JDL [6] dokumentumot, melynek segítségével eldönti, hogy milyen erőforrásokra van szüksége az adott jobnak, és ezt melyik CE biztosítja a legoptimálisabban.

Információs rendszer

Minden esetben az információs rendszer [1] a GRID lelke, mivel ismernünk kell a GRID komponenseire vonatkozó információkat ahhoz, hogy a jobok ütemezésekor, illetve a GRID irányítása esetén a megfelelő döntéseket tudjuk meghozni.

A rendszer egy egységes módszert biztosít a komponensekre vonatkozó információk megszerzésére.

Az információs rendszer statikus és dinamikus adatokat szolgálhat. A statikus adatok esetében nincs, vagy ritka a változás, így itt nem kell az adatok elavultságával számolnunk. Ehhez a kategóriához tartoznak például egy adott gép processzorának típusa. A dinamikus adatok esetében viszont nagy probléma az aktualitás biztosítása, mivel ezek folyamatos változásban vannak. Ebből következik, hogy a rendelkezésünkre álló dinamikus információk mindíg elavultak, mire a rendszeren keresztül eljutnak a felhasználóhoz.



2.2. ábra. A Production GRID-ek elvi felépítése.

Site

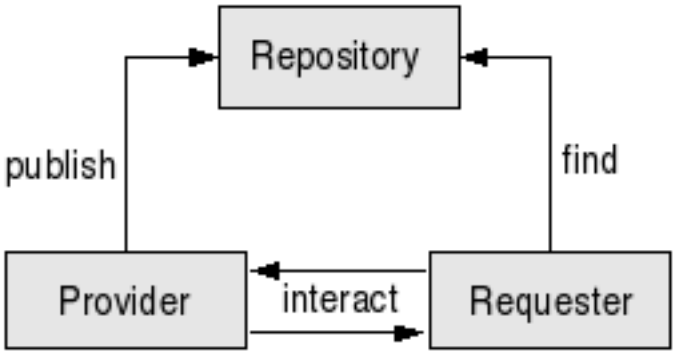
A GRID legnagyobb fizikai egysége, melyben SE-ek és CE-ek találhatók és a globális hálózatnál gyorsabb hálózati kapcsolattal vannak összekötve. Az optimalizálás szempontjából fontos egység, mivel a site-on belüli kommunikáció gyorsabb, mint ha ezt a globális hálózaton tennénk. Általában egy intézmény által fentartott komponensek tartoznak egy Site-hoz.

A rendszer szerkezete

A rendszer több rétegre osztható. A globális GRID-et Site-okra oszthatjuk, mely azt mutatja, hogy mely komponensek találhatók egy lokális hálózaton belül. A Site-on belül elhelyezhetünk CE-eket és SE-eket. A CE pedig egy klaszter szintű lokális ütemezést végez(2.2 ábra).

2.1.5. Globus Toolkit 2 (GT2) alapú GRID-ek

Az első jelentős middleware a GT2 [7] volt, melyre a jelenleg működő Production GRID-ek legtöbbje épül.



2.3. ábra. Szolgáltatások igénybevétele.

A GT2-re épülő rendszerek komponenseinek cseréje nehezen kivitelezhető. A stabilitás érdekében előre meghatározott operációs rendszerre meghatározott programcsomagokat kell telepíteni és bekonfigurálni. Az architektúra viszonylag statikus.

A GT2 által bevezetett biztonsági rendszer általánosan elterjedt megoldást biztosít a GRID rendszer komponenseinek védelmére és a felhasználók hozzáféréseinek kezelésére.

2.1.6. Szolgáltatás-alapú GRID

A jelenleg bevezetés alatt álló GRID rendszerekben jelentős szemléletbeli változást jelent, hogy mindent szolgáltatásnak tekintenek. Ez azt jelenti, hogy mindent valamilyen szolgáltatás mögé helyeznek. Ennek következtében az absztrakció miatt könnyen módosíthatóak a rendszer komponensei, és bármikor új szolgáltatást vezethetnek be. A rendszer minden komponensét egységesen kezelhetjük, mivel mindegyik egy szolgáltatásként jelenik meg.

A rendszer biztosítja a szolgáltatások felderítédét és adminisztrálását.

A szolgáltatás igénybevételénél három szereplő jelenik meg. Az első a szolgáltató, a második a szolgáltatást igénybe vevő, míg a harmadig a *Repository*, mely a szolgáltatás felderítését biztosítja (2.3 ábra). Első lépésként a szolgáltató feljegyezteti magát a *Repository*-ban, majd az ügyfél a számára legmegfelelőbb szolgáltatást kiválasztja a *Repository* segítségével. Ezek után az ügyfél közvetlenül igénybe veheti a szolgáltató által nyújtott szolgáltatást.

A szabványos kommunikáció eredménye képpen az ügyfél lehet egy másik szolgáltatást nyújtó szolgáltató is. Ebből láthatjuk, hogy a szolgáltatások más szolgáltatások segítségével is létrejöhetnek, mely nagy mozgásteret biztosít a middleware tervezőinek és karbantartóinak.

A GRID rendszerek esetében fontos a szolgáltatók dinamikus létrehozása

és megszüntetése. A szolgáltatás-alapú GRID rendszerek biztosítják a lehetőséget, hogy bárki létre hozhasson egy szolgáltatót az adott gépen elhelyezett *Factory* segítségével.

A szolgáltató igény szerint jön létre, majd a használat után megszűnik. Ennek eredménye képpen biztosíthatjuk, hogy ne maradjanak feleslegesen futó programok a használat befejezésekor.

A szolgáltatás-alapú megközelítés jelentős előrelépést jelenthet a biztonság tekintetében is, mivel ha szolgáltatást veszünk igénybe, akkor az adott erőforráson nem a saját kódunkat futtatjuk, hanem egy jól tesztelt és biztonságos szolgáltatáshoz férünk hozzá.

A szolgáltatás-orienált megközelítést megvizsgálhatjuk különböző szempontokból.

- Logikai szempontból: A szolgáltatás egy absztrakt, logikai nézete különböző programoknak, adatbázisoknak vagy üzleti folyamatoknak.
- Üzenetek szempontjából: A szolgáltatás a szolgáltató és az ügyfél közti lehetséges üzenetek segítségével van definiálva. Ennek segítségével a szolgáltatás megvalósítása és felépítése ismeretlen maradhat a külvilág elől.
- A leírás szempontjából: A szolgáltatás leírása más programok által értelmezhető meta-adatok formájában történik.
- Műveletek szempontjából: A szolgáltatások viszonylag kis számú műveletet biztosítanak, melyeket nagy és bonyolult üzenetekkel írhatunk le.
- A hálózat szempotjából: A szolgáltatások úgy vannak kialakítva, hogy hálózati kommunikáció segítségével lehessen igénybe venni őket.
- A platformfüggetlenség szempontjából: Az üzeneteket szabványos és platformfüggetlen formában küldjük.

2.1.7. Globus Toolkit 4 (GT4)

A GT4 [7][8] a világ legelterjedtebb eszköze a webszolgáltatás alapú GRID rendszerek felépítésére. Bizonyos szintű kompatibilitást biztosít a korábbi verziókkal.

A GT4 szoftver-komponenseket biztosít elosztott rendszerek építésére webszolgáltatások segítségével. A webszolgáltatás egy megoldást biztosít különböző szoftver-komponensek együttműködésére hálózati kommunikáció segítségével.

A szolgáltatás interfészét a WSDL [28] segítségével írhatjuk le. Más komponensek SOAP [27] üzenetek segítségével érhetik el a szolgáltatás interfészét. A SOAP üzeneteket HTTP protokoll segítségével szállíthatjuk a komponensek között. Az üzenet leírását XML [5] formátumban adhatjuk meg. Ebből látszik, hogy a kommunikációhoz szabványos protokollokat alkalmazunk. Ennek következtében platformfüggetlen szolgáltatásokat hozhatunk létre.

Magasabb szinten szükség van a biztonság és a szolgáltatások menedzseléséhez szükséges specifikációkra is.

A WS-Security specifikációban leírtak biztosítják a szolgáltatások biztonságos igénybevételét. Ennek elemei az authentikáció, authorizáció, jogosultságok reprezentálása és a partner kilétének felismerése.

A WS-Addressing írja le a webszolgáltatások címzését. A WSRF[26] biztosítja a webszolgáltatások menedzselését. A WS-Notification pedig a webszolgáltatások közti információcserére nyújt megoldást.

Ezen specifikációk segítségével készültek a GRID rendszerek felépítéséhez szükséges komponensek.

Ezeket több csoportra oszthatjuk.

- Alapvető komponensek: Ezek biztosítjá a webszolgáltatás készítéséhez szükséges elemeket (Java WS Core, C WS Core, Python WS Core), valamint a fájlok elérésének egy absztrakcióját(XIO).

- Biztonságért felelős komponensek: Ezen komponensek biztosítják az authorizációt, authentikációt, delegációt, credential menedzselését és az accounting lehetőségét.

- Adatelérést biztosító komponensek: E komponensek valósítják meg a replika katalógust, valamint biztosítják a fájlok elérését.

- Információs szolgáltatások: Webszolgáltatás segítségével megvalósított információs rendszer(MDS4), mely biztosítja a rendszerben található szolgáltások felderítését és a rájuk vonatkozó információkat.

- Futtatás menedzselésért felelő komponensek: Webszolgáltatások segítségével valósítják meg a jobok futtatását.

- Komponensközi eszközök: Ezek magasabb szintű keretrendszert biztosítanak a GRID használatához.

E komponensek egy alapot biztosítanak a GRID rendszer összeállításához. Könnyedén illeszthetünk bele újabb komponenseket, aminek segítségével az igényeinknek és elképzeléseinknek megfelelő GRID rendszert hozhatunk létre.

2.1.8. gLite

A GT4 fejlesztésével párhuzamosan készíti az EGEE [9] a saját szolgáltatás-alapú GRID middleware-jét, melyet gLite-nak [17] hívnak. Ez a rendszer szintén a szolgáltatás alapú megközelítésre épít. A GT4-gyel ellentétben egy teljes rendszert állítottak össze, melynek feltelepítésével szinte működő GRID rendszert kaphatunk.

Európában jelenleg is folyik a rendszerre való áttérés.

A rendszer szerver és kliens oldali komponensekből áll. Általában minden szerver oldali komponenshez készítettek egy kliens oldali API-t, melynek segítségével könnyedén igénybe vehető a szerver által nyújtott szolgáltatás.

A rendszer lelkét az információs rendszer képezi (R-GMA [18]). A kompatibilitás megőrzése érdekében képes a korábbi LDAP [19] segítségével tárolt GLUE [20] sémának megfelelő adatok tárolására is.

Az információs rendszer egy virtuális relációs adatbázis, melyhez a jól ismert *SQL* parancsok segítségével férhetünk hozzá.

A GRID rendszerben található szolgáltatások regisztrálhatják magukat a rendszerben, hogy más komponensek is tudjanak a létezésükről.

Az információs rendszerhez több programozási nyelven is létezik API, melyek biztosítják a különböző parancsok kényelmes végrehajtását.

2.1.9. Multi-ágens alapú GRID

A GRID rendszerek következő generációjának alapkövei az ágensek lehetnek. Ebben a megközelítésben minden komponenst ágensek képviselnek. Egységes nyelvet beszélve képesek egymással kommunikálni. Képesek a helyváltoztatásra, hogy a szükséges tevékenységeket a megfelelő helyen végezzék el. Itt a döntéseket nem egy központi RB hozza meg, hanem a képviselettel felruházott ágensek. Egy vásártérnek(Marketplace) nevezett helyen találhatják meg a számukra szükséges erőforrásokat, majd a tárgyalás után megállapodnak a használat feltételeiről.

A skálázhatóság szempontjából is fontos, hogy nem központilag hozzuk meg a döntéseket, hanem az ágensek által a döntéshozás is elosztottá válik.

Ezen architektúra segítségével létrehozhatunk egy szemantikus GRID rendszert.

Ebben a rendszerben lehetőség van, hogy saját magunk által készített ágensek képviseljenek bennünket. Ennek eredménye képpen tetszőleges stratégiát használhatunk a szükséges erőforrások kiválasztására, valamint az erőforrások is tetszőleges ágenssel képviseltethetik magukat.

A jelenlegi rendszerekben nagy problémát jelent, hogy nincs kielégítő módszer a szolgáltatások értékesítésére. Az ágens-alapú GRID esetében egy

piachoz hasonlóan a feleket képviselő ágensek megegyezhetnek a használat díjáról.

Az ágensek képesek egymással kommunikálni. A kommunikáció alapvetően aszinkron módon történik üzenetek küldésének segítségével. Egy absztrakciós szint bevezetésével lérejön egy absztrakt tér, melyben a felek az üzenetet eljuttathatják egymásnak.

2.2. GRID alkalmazások

2.2.1. Számításigényes alkalmazások

Egy számításigényes probláma esetén az alkalmazás a futás közben viszonylag kis mennyiségű adatot használ, de azon hosszú ideig tartó számításokat végez. Ebben az esetben nem jelent problémát a fájlok elérése, mivel az ehhez szükséges idő nagyságrendekkel kisebb a processzor által végzett számításokra fordított időnél.

Ezeknél az alkalmazásoknál a legfontosabb probléma, hogy hiba esetén a már elvégzett akár napokig vagy hetekig tartó számításokat ne kelljen elölről kezdeni.

2.2.2. Adatintenzív alkalmazások

Az adatintenzív alkalmazások számunkra érdekesebbek. Ezen alkalmazások esetében a feladat, hogy nagy mennyiségű adatot dolgozzunk fel. Itt arra van szükség, hogy az adatok elérése a lehető leggyorsabb legyen, mivel ennek sebessége alapvetően meghatározza az alkalmazás futásához szükséges időt.

Azoknál a kutatási területeknél, ahol valamilyen mintavételezés után szeretnék feldolgozni a keletkező adatokat, melyek mérete akár több terabájt is lehet, legtöbbször ilyen típusú alkalmazásokat alkalmaznak.

2.2.3. Szekvenciális alkalmazások

Más szempontból vizsgálva, megkülönböztethetjük az alkalmazásokat az alapján, hogy hány processzort használnak a futásuk közben.

A legegyszerűbb eset, ha egy szekvenciális programot szeretnénk végrehajtani a GRID egy WN-ján. Ebben az esetben nem kell figyelembe vennünk a GRID többi részének állapotát, vagy a futtatásra használt klaszter szabad WN-jainak számát.

A jobhoz kiválasztunk egy WN-ot, majd végrehajtjuk az alkalmazást.

2.2.4. Párhuzamos alkalmazások

Abban az esetben, ha a végrehajtandó job egy párhuzamos program, akkor fontos, hogy milyen módon választjuk ki a végrehajtáshoz használt klasztert.

Ebben az esetben a jobhoz több WN-ot is le kell foglalnunk. Meg kell vizsgálni, hogy van-e lehetőség egy klaszteren belül lefoglalni ezeket az erőforrásokat, vagy a párhuzamos program processzei különböző klaszterekbe is kerülhetnek. Ez utóbbi esetben biztosítani kell, hogy a két klaszter között lehetőség legyen a kommunikációra. Azt is figyelembe kell vennünk, hogy olyan processzek melyek nagy mennyiségű kommunikációt folytatnak, ne kerüljenek messze egymástól.

Párhuzamos programok készítésére alapvetően két eszközt használnak, melyek a PVM [14] és az MPI [15].

Ezek az alkalmazások azonban a GRID szempontjából egy alkalmazásnak tekinthetők, csak a végrehajtásukhoz több erőforrásra is szükség van.

2.2.5. Workflow alkalmazások

A következő lépést a workflow alkalamazások jelentik, ahol több job végrehajtásáról beszélhetünk, melyek a GRID szempontjából szinte független alkalmazásoknak tekinthetők, de a részfeladatok eredményei befolyásolják az utánuk következő részfeladatok végrehajtását.

A workflow alkalmazásnak megfeleltethetünk egy irányított körmentes gráfot(DAG), melynek a csomópontjaiban jobok találhatók, és az élei a köztük lévő függőségeket reprezentálják. Ha két csomópont között él van, akkor az él kiindulási csúcsa olyan adatokat állít elő, melyre a célcsúcsnak szüksége van. A kiindulási csúcsban található job által előállított adatokat el kell juttatni azoknak a csúcsoknak, melyeknek szüksége van rá. Ezen alakalmazások nagy jelentőséggel bírnak, mivel számos problémát egy workflow segítségével oldhatunk meg. Minden rendszerben nagy figyelmet szentelnek az ilyen típusú alkalmazások lehetőségének.

Az is lehetséges, hogy az egyik csomópontban található részjob egyben egy párhuzamos program, melynek végrehajtásához több WN használatára is szükség lehet.

A workflow alkalmazások kezelésére több rendszert is kidolgoztak. Egy ilyen például a P-GRADE [16] portál, amely lehetővé teszi a felhasználók számára, hogy könnyedén készítsenek workflow alkalmazásokat, és azok részfeladatait akár különböző GRID-ekben is végrehajthassák. A portál lehetővé teszi, hogy az egyes részjobok akár párhuzamos programok is legyenek, valamint megkíméli a felhasználót attól, hogy a parancssori utasításokat kelljen

alkalmaznia, melyek különböző GRID-ek esetén teljesen mások is lehetnek.

2.3. Fájlelérés a GRID rendszerekben

Az adatintenzív alkalmazások szempontjából döntő fontosságú a fájlok elérése. Ezen alkalmazások nagy mennyiségű adathoz kell, hogy hozzáférjenek. Az adatelérés gyorsításával jelentős előnyhöz juthatunk a futtatási idő tekintetében.

A Production GRID-ek egyik komponense a SE, mely az adatok tárolását teszi lehetővé. A jelenlegi GRID rendszerekben azt a módszert alkalmazzák, hogy minden Site-on található legalább egy SE. Az adatfájlokat csak létrehozáskor lehet módosítani. Ezek után a GRID-fájl csak olvasható. Ezen megszorítás eredményeképpen lehetőség van a fájl tőbb példányban való másolására úgy, hogy biztosak lehetünk a tartalom azonosságában. Ez a megszorítás a legtöbb esetben nem jelent nagy problémát, mivel ritkán van szükség az adatok módosítására. Ebben az esetben azonban jelentősen lassabb ez a megoldás, mivel a fájl változatlan részét le kell másolnunk.

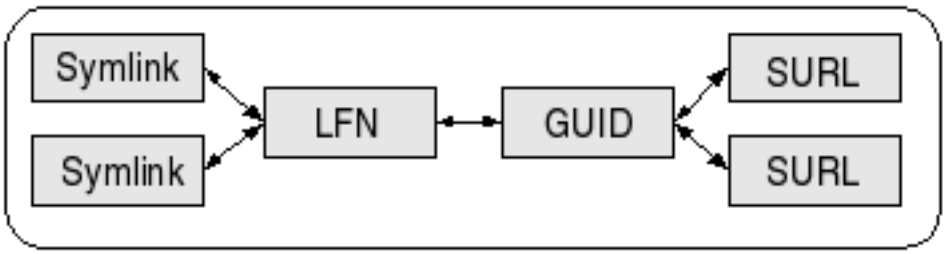
2.3.1. LCG-2 GRID

Az LCG-2 [10] GRID esetében a következő módon történik a fájlok elérése. A GRID-fájlok megtalálhatóak lehetnek több Site SE-jén is. Ha egy olyan jobot küldünk a GRID-be, amelynek szüksége van az adott GRID-fájlra, akkor a RB először megvizsgálja, hogy melyik CE Site-ján található meg a fájl, és csak ezután veszi figyelembe a többi feltételt. Így biztosítja, hogy a fájlok elérése a Site-on belöl történjen. Ennek következtében a CE és a SE közti kommunikáció a Site-on belüli gyors lokális hálózaton történik.

Ebben a GRID-ben a Site-on kívüli SE-k elérése csak a GridFTP [11] segítségével történhet. A Site-on belüli elérésre azonban rendelkezésre áll több API is.

A fájlok felderítésére ezekben a rendszerekben egy fájlkatalógust használhatunk, melyben nyilvántartásba kerülnek a GRID-fájlok. Az egyértelmű azonosításra a GUID szolgál, mely a GRID-fájl létrehozásakor generálódik, és biztosítva van az egyedisége. Egy GUID-hez számos másolat tartozhat és több logikai fájlnévvel(LFN) vagy szimbólikus linkkel hivatkozhatunk rá, mely biztosítja a kényelmes azonosítást. Az adatbázisban a GUID-hez hozzá vannak rendelve a GRID-fájl másolatainak eléréséhez szükséges alacsonyabb absztrakciós szinten található címek(SURL) is (2.4 ábra).

Ezen információk elérésére és módosítására használhatók parancssori alkalmazások, vagy a rendszerhez készített API-k.



2.4. ábra. A fájl- és replika-katalógus felépítése az LCG-2 GRID-ben.

Job futtatásakor végzett műveletek

Ebben a rendszerben egy job futtatása esetén, ami a leggyakoribb és számunkra legfontosabb eset, két mód van a fájlok elérésére.

A első és legegyszerűbb módszer, hogy a kis méretű fájlokat a futtatható fájl mellett átmásoljuk a futtatás helyére, majd a kisebb kimeneti fájlokat a futás befejeztekor elérhetővé teszük a futtatást kezdeményező felhasználó számára. Ez a módszer csak a kis méretű fájlok esetén hatékony.

A második eset, hogy egy SE által tárolt fájlt használunk. A JDL nyelv lehetőséget ad a GRID-fájlok felsorolására. Ebben az esetben az ütemező elsődleges szempontnak tekinti, hogy olyan CE-t válasszon, melynek közelében megtalálhatóak az adott fájlok másolatai. Ez biztosítja a fájlok elérésének optimalizálását. Ezen GRID-fájlok olvalását nem végzi el a rendszer. Ezt a jobnak valamely API segítségével kell megtennie. A job lefutása után létrejön a kimeneti fájl. Ennek létrehozását elvégezhetjük az API-k segítségével is, de ebben az esetben nem kell megadnunk a JDL fájlban kimeneti fájlként. Abban az esetben, ha a WN lokális tárolóján hozzuk létre a kimeneti fájlt, melyet GRID-fájlként szeretnénk tárolni, meg kell adnunk kimeneti fájlként ezt a fájlt a JDL leírásben. Ennek hatására a rendszer autómatikusan létrehozza az adott nevű GRID-fájlt és az adatokat felmásolja a megadot SE-re a job befejezése után.

2.3.2. gLite

Az EGEE gLite rendszerében, amely már a szolgáltatásokra épül, kidolgoztak egy szolgáltatást, mely lehetővé teszi a fájlok elérését egy jól meghatározott interfészen keresztül. Ez a komponens egy szerver oldali alkalmazásból és egy kliens oldali API-ból áll. Lehetőség van a szerverek láncolására, amivel kihasználhatjuk a cache-elés adta optimalizálási lehetőségeket. Az API biztosít egy POSIX interfészt is a fájlok elérésére, melynek segítségével a hagyományos lokális fájlokhoz hasonlóan hozhatjuk létre és olvashatjuk a

fájlokat.

2.4. Optimalizálás

A job lefutását úgy tehetjük gyorsabbá, hogy az IO műveleteinek végrehajtási idejét csökkentjük. Ennek több szintje van.

A jelenlegi rendszerek arra törekszenek, hogy a szükséges adatok a jobot futtató WN-dal egy Site-on helyezkedjenek el. Ennek következtében az adatok elérése nem a viszonylag lassú globális hálózaton történik, hanem a Site nagyságrendekkel gyorsabb belső hálózatán.

Jelentősen gyorsíthatjuk a végrehajtást, ha az adatokat a WN lokális háttértárolójára másoljuk mire szükség lenne rá. Ezzel a módszerrel az adatok eléréséhez szükséges idő jelentősen csökkenthető.

Feltételezhetjük, hogy ismerjük, hogy a jobnak mely fájlok mely részeire van szüksége [12][13]. Ezen információ segítségével csak a szükséges adatokat kell a lokális háttértárra másolni.

2.4.1. Adatok másolása és a job életciklusa

Megvizsgálhatjuk, hogy a job futása és a számára szükséges adatok másolása mikor történhet. A jelenlegi rendszerekben általában, vagy a job indítása előtt másolódik le az egész bemeneti fájl, vagy a job futása közben egy távoli SE-en éri el. Az első esetben az a probléma, hogy nincs arra lehetőség, hogy a fájloknak csak bizonyos részeit másoljuk le. A második esetben pedig a fent már említett kommunikációs költséggel kell számolnunk.

Az első javítási lehetőség, hogy a job indításával egyidőben kezdjük letölteni a szükséges fájlrészleteket. Ennek segítségével a job futásával párhuzamosan, és várhatóan a kérés előtt tölthetjük le az adatokat.

Egy másik lehetőség, hogy a job a WN-ra érkezése előtt egy ágenst küldünk, hogy kezdje el az adatok másolását. Ebben az esetben a nagyobb fájlok másolása is befejeződhet a kérés megtörténte előtt, valamint csökkentjük a futással párhuzamosan történő letöltéssel járó terhelést. Hátránya, hogy egy másik job futása alatt terheljük az erőforrást.

2.4.2. Workflow alkalmazások optimalizálása

Egy workflow alkalmazás esetében jelentősen csökkenthetjük a végrehajtáshoz szükséges időt, ha a helyzetnek megfelelő módszert választjuk az adatok elérésére.

A workflow olyan csomópontjaiban, melyeknek olyan adatra van szüksége, ami a workflow futtatása előtt rendelkezére áll, alkalmazhatjuk az előzőekben javasolt ágenssel történő előreolvasást.

Akkor is gyorsíthatunk, ha egy csomópontban előállítunk egy ideiglenes fájlt, melyre más csomópontoknak van szüksége. Ha az ideiglenes fájlt szekvenciális módon hozzuk létre és a rákövetkező csomópontokban szekvenciálisan dolgozzuk fel, akkor jelentősen gyorsíthatjuk a futást. A jelenlegi esetben, vagy egy SE-n tároljuk az ideiglenes fájlt, vagy a létrehozó csomópont lefutása után kerül átmásolásra.

Egy lehetőség, hogy a rákövetkező csomópont futtatásának helyére egy ágenst helyezünk, mely a fájlt előállító csomóponttól a futási idővel egyidőben letölti az előállított adatokat. Ezzel a módszerrel szinte a létrehozó jobbal egyidőben a fájl a szükséges WN-okra kerülhet.

Ennél tovább is léphetünk, ha a feldolgozást végző csomópontokban előbb indítjuk el a jobot mint, hogy a megelőző csomopontok befejeződnének. Így a párhuzamos programozásban ismert adatcsatornák modelljéhez hasonló eredményt kaphatunk.

3. fejezet

Elemzések

3.1. Szolgáltatás és ágens alapú megközelítések összehasonlítása

A két megközelítés esetén más szempontok szerint történt a rendszerek elveinek kidolgozása. E miatt meg kell vizsgálni, hogy melyik rendszer milyen feladatok megoldására alkalmas.

3.1.1. A szolgáltatás

A szolgáltatás alapvetően az objektum-orientált megközelítéshez hasonló elvekre épül. Ez alatt azt értem, hogy adott egy szolgáltatás, melynek adva van egy jól definiált interfésze. Ez egy absztrakciós szintet jelent. Egy másik komponens igénybe vehet az interfész által leírt szolgáltatásokat. Ebben az esetben az interfészen keresztül kérheti a komponenst, hogy bizonyos műveletek végrehajtása után visszaadja a kért művelet eredményét.

Ebben az esetben egy egyirányú kapcsolatot épít fel a szolgáltatást igénybe vevő a szolgáltatást nyújtó komponenssel. Ennek következtében a válasz küldéséhez nincs szükségünk a kliens oldali komponens felé kapcsolat kiépítésére. Ez biztonsági szempontból előnyt jelent.

3.1.2. Az ágens

Az ágens kidolgozásánál az elsődleges szempont a kommunikáció volt. Ennek érdekében úgy alkották meg a szabványokat, hogy bármely ágens bármely ágenssel "beszélgethessen". Ebből adódóan az ágensek közti kommunikáció aszinkron módon történik. Az ágensek mindegyike egy egyértelmű azonosítóval rendelkezik. Ha egy ágens kommunikációt szeretne folytatni egy

másik ágenssel, akkor az egyértelmű azonosítójának az ismeretében az absztrakt téren keresztül egy üzenetek küldhet neki. A másik ágens az üzenetből kinyerheti a partner azonosításához szükséges adatokat. Az üzenet hatására bizonyos állapotváltozásokat végezhet. Az állapotváltozás következtében lehetséges, hogy válaszként üzenetet küld a másik félnek. Ebből látszik, hogy ebben az esetben az ágensek kommunikációja bizonyos protollokon keresztül történik.

A fentiekből következik, hogy mindkét félnél szükséges az, hogy kapcsolatot tudjon kiépíteni a partnerrel, tehát a rendszer minden komponensének nyitottnak kell lenni a kommunikációra. A szolgáltatásokkal ellentétben itt a csak kliens szerepet betöltő komponensekhez is kapcsolatot kell kiépítenünk.

Az ágensekben található egy sor, mely viselkedéseket tartalmaz, és az ágens végrehajtja a soron következő viselkedést. Ennek segítségével a megfelelő eseményeknek megfelelő viselkedés kerülhet végrehajtásra. A valódi tevékenységet a viselkedések által valósíthatjuk meg.

3.1.3. Szolgáltatások rendszerének felépítése

A szolgáltatások rendszerének a magját a *Repository* jelenti. Ez szolgáltatja az információkat a rendszer komponenseit jelentő szolgáltatásokról. Lehetőséget biztosít, hogy egy szolgáltatás eléréséhez szükséges információkhoz hozzájussunk, valamint felderítsük a szolgáltatás igénybevételéhez szükséges interfészt. Ezen információk birtokában kapcsolódhatunk a szolgáltatóhoz, és automatikusan generálhatjuk az interfésznek megfelelő üzeneteket.

Láthatjuk, hogy a szolgáltatások esetében egy kliens és egy szerver közti kommunikációból indulunk ki. Ebből adódik, hogy a rendszer alapvetően hierarchikus felépítésű. A kliensek szerverek szolgáltatásait vehetik igénybe, de egyben lehetnek szerverek is, melyet más kliensek használhatnak.

Természetesen van lehetőség arra, hogy a kapcsolatok gráfjában köröket alakítsunk ki, vagy akár egy egyenrangú felek közti kommunikációt is szimulálhatunk. Azonban a rendszerre alapvetően ez a hierarchia jellemző.

A szerver-kliens felépítés eredménye képpen az objektum-orientált világban járatos tervező és fejlesztő szakemberek számára a szolgáltatás-alapú komponensek megalkotása nem jelent problémát. Adott egy jól definiált interfész. Ezen interfésznek megfelelően könnyedén megvalósítható az adott komponens.

A szolgáltatásokra épülő GRID rendszerek jelenleg már működésben vannak. Ki vannak dolgozva a szolgáltatások készítéséhez szükséges eszközök. Meg vannak teremtve a kényelmes és egyszerű fejlesztés feltételei. Rendelkezésre állnak a megfelelő API-k, melyek segítségével könnyen és megbizhatóan készíthetjük el a megfelő szolgáltatásokat.

A szolgáltatás igénybevételéhez pedig automatikusan generálhatjuk a szükséges API-kat, ami nagy segítséget jelent a kliens oldali komponensek fejlesztésében.

3.1.4. Ágensek rendszerének felépítése

Az ágensek rendszerében az üzenetküldést az absztrakt tér megvalósítása végzi. A partner azonosítójának ismeretében az absztrakt tér biztosítja, hogy az üzenet a megfelelő komponenshez jusson el. A megfelelő tulajdonságokkal rendelkező ágens felderítéséhez azonban itt is igénybe kell vennünk egy információs rendszert, melybe az ágensek feljegyezhetik magukat. Tehát ebből a szempontból a két rendszer felépítése megegyezik.

A legjelentősebb különbség abban van, hogy alapvetően nincs egy hierarchikus felépítés a komponensek között. Minden komponens egyenlő, és bárki kommunikálhat bárkivel. A rendszer egy emberekből álló rendszerhez hasonlítható. Alapvetően minden szereplő egyenlő, de a szerepeiknek megfelően kialakulhatnak a kliens-szerver szerű kapcsolatok. Láthatjuk, hogy vannak olyan ágensek a rendszerben (például az információs rendszert megvalósító ágens), melyek egy szolgáltatást valósítanak meg. Ebből megállapíthatjuk, hogy a rendszer felépítésének tekintetében az ágens-alapú megközelítés egy általánosítása a szolgáltatásokra épülő rendszereknek. Abból a szempontból nyújtanak többet az ágensek, hogy lehetőség van egyenrangú tárgyalófelek megvalósítására, melyek üzenetek küldésével képesek egy bizonyos cél érdekében megállapodást kötni.

A tárgyalásos megállapodás következtében számos lehetőség merül fel a rendszer működésének fejlesztése terén.

Ennek segítségével a központilag meghozott ütemezési eljárásokat a piacgazdasághoz hasonlóan elosztott módon valósíthatjuk meg [30]. Az ágensek közti megállapodások segítségével történhet az ütemezés. Ennek következtében a rendszer skálázhatósága nagyságrendekkel növelhető.

A piacgazdaság működéséhez hasonlóan megoldható az erőforrások piaci értékesítése. A felek megállapodásának megfelelően történhet az erőforrások használatának számlázása. Ez nagyon fontos előrelépést jelenthet a GRID rendszerek fejlődésében, mivel a jelenleg használatos rendszerek alapvetően kommunisztikus elven működnek, vagyis a különböző szervezetek felajánlanak erőforrásokat, melyet a felhasználók ingyen használhatnak. A gazdasági szereplők bevonásával nagy lépést lehetne elérni, de ennek érdekében szükség van arra, hogy haszonszerzési lehetőséget jelentsenek ezek a rendszerek.

Az ágens alapú rendszerek kidolgozása jeleneleg is folyik. A FIPA [23] a multi-ágens rendszerek szabványosító szervezete. A szabványok alapján fejlesztik a JADE [21][22] rendszert, mely Java nyelven íródik és megteremti a

lehetőséget egy ágens alapú rendszer kiépítésére.

Ha szeretnénk kiépíteni egy ágensek alkotta GRID rendszert, akkor meg kell vizsgálnunk, hogy milyen típusú komponensekre van szükség. Szükségünk van a szolgáltatást nyújtó ágensekre, amely esetben egy kliens-szerver kapcsolat jelenik meg. Ekkor a jelenlegi eszközökkel való fejlesztés nagyon nehézkes, ezért fontos lenne egy olyan lehetőség kidolgozása, mely egy interfész megvalósítását teszi lehetővé. Tehát legyen lehetőség arra, hogy egy adott interfészhez automatikusan létre lehessen hozni az ágens szintű kommunikációt végző viselkedést.

Ebben az esetben az ágensek felhasználásával szimuláljuk a szolgáltatásokat úgy, hogy egy absztakciós szinttet vezetünk be.

Az ágens-alapú kommunikáció esetében nem egy szabványos alacsony szintű protokol van, mely biztosítaná az üzenetek továbbítását, hanem több lehetőség áll rendelkezésre. Például lehetőségünk van e-mail segítségével célba juttatni az üzenetet. Ennek jó kihasználásához biztosítani kell azt az információt, hogy melyik ágens milyen szállítási protokollon képes fogadni az üzeneteket.

3.1.5. Szolgáltatás-alapú GRID felépítése

A szolgáltatások jelentik a rendszer építőköveit. Minden komponenst szolgáltatások valósítanak meg. A rendszer lelke az információs szolgáltatást megvalósító komponens, mely lehetővé teszi a szolgáltatások felderítését. A GRID rendszerben való alkalmazáshoz vannak igazítva a web-szolgáltatások, vagyis a szolgáltatások dinamikusan jönnek létre és szünnek meg, míg a web világában ez statikusan történt. Ennek eredménye képpen fontos a szolgáltatások életciklusának kielégítő módon való kezelése, és az információk elavultságának csökkentése a lehető legkisebb szintre. Ez utóbbi probléma megoldására bevezették annak a lehetőségét, hogy bizonyos változások esetén értesítést küldhessünk azoknak a komponenseknek, melyek előzőleg igényelték ezt.

3.1.6. Ágens-alapú GRID felépítése

Az ágens alapú GRID esetében a legfontosabb, hogy minden szereplőt ágensek képviselnek. Ágensekkel valósítjuk meg a szolgáltatásokat, ezek képviselik a felhasználók érdekeit, valamint tárgyalnak az erőforrások érdekében.

A tárgyalás színhelye a *piactér*. Az érdekeket képviselő ágensek a piactéren találkoznak. Az itt összegült képviselők valamilyen mahanizmus alapján tárgyalásokat folytatnak, majd megállapodnak az erőforrások használatának

feltételeiről. A megállapodás mechanizmusa lehet valamilyen árverési rendszer, mely a piaci életben jól kidolgozott és bevált módszert biztosít az értékesítésre.

Fontos elem, hogy az ágensek képesek a helyváltoztatásra, vagy más ágensek távoli helyen való futtatására. Ennek kihasználásávan küldhetjük a képviselőnket a *piactérre*, vagy lehetőségünk van egy ágens egy adott erőforráshoz való elküldésére, mellyel elvégezhetjük a szükséges előkészületeket.

A fentiek azonban biztonsági szempontból számos problémát jelenthetnek. Ha bárhova küldhetünk ágenseket, akkor biztosítanunk kell, hogy azok ne tehessenek bármit. A visszaélések problémát jelenthetnek.

A jobok esetében meg volt teremtve a feltétele annak, hogy egyszerre ne használjon egy erőforrást több job, mivel az ütemező gondoskodott a jobok indításáról.

Abban az esetben, ha egy előkészítő ágenst küldünk egy erőforrásra, akkor biztosítani kell, hogy csak egy minimális szintig terhelhesse az erőforrást, mivel ha annak használatát egy másik felhasználó az adott időtartamra megvásárolta, akkor megkárosíthatjuk az erőforrás jogosulatlan eltulajdonításával.

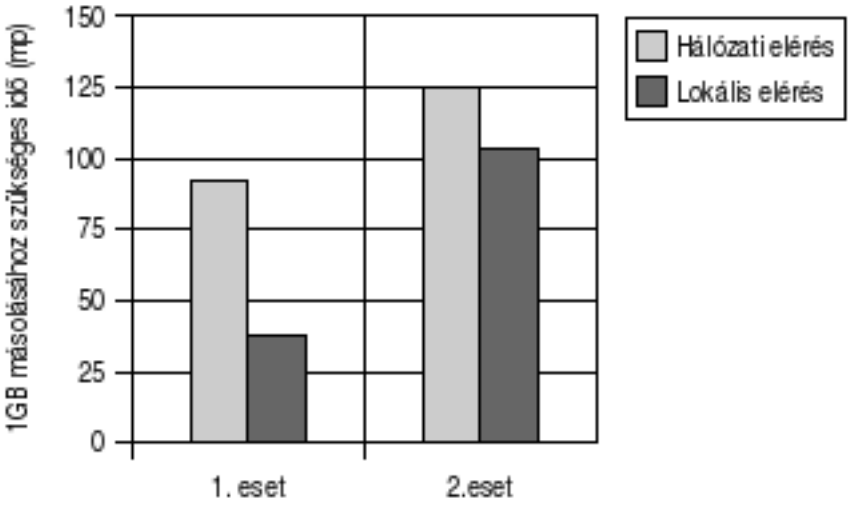
Ahogy a szolgáltatások esetén, itt is fontos megteremteni a jobok futtatásának lehetőségét, ami azonban nem illeszkedik teljesen ezen rendszerek felépítésébe. Ennek érdekében a szolgáltatások esetében használt megoldáshoz hasonlóan szükség van egy olyan szolgáltatást megvalósító ágens alkalmazására, mely lehetővé teszi a job futtatását.

3.1.7. Összehasonlítás

Láthattuk, hogy az ágensek által nyújtott lehetőségek sokkal bővebbek a szolgáltatásoknál, azonban a szolgáltatások készítésének és használatának feltételei ki vannak dolgozva, és a GRID rendszerekben is alkalmazható eszközök állnak rendelkezésre.

Az ágensek esetében még nem tartunk itt. A GRID rendszerekben való alkalmazáshoz meg kell vizsgálni a megoldásokat a skálázhatóság és a sebesség tekintetében. Kényelmessé kell tenni a kliens-szerver alkalmazások fejlesztését, melyek a rendszer tervezésében fontos szerepet játszhatnak. Át kell gondolni a kommunikáció kérdését, hogy a legjobb megoldást találjuk meg a GRID rendszerekhez.

A kommunikáció tekintetében az ágensek rendszere nem hasonlít a szolgáltatások rendszeréhez. Ennek oka a kétirányú kapcsolat szükségessége, melynek alkalmazása a biztonság tekintetében visszalépést jelenthet a jelenlegi rendszerekhez képest.



3.1. ábra. A hálózati és a lokális fájlelérés összehasonlítása

3.2. A lokális háttértár használatának elemzése

Annak érdekében, hogy meggyőződjek arról, hogy valódi előnyt jelent, ha a WN a lokális háttértárolójáról olvassa az adatot, néhány mérést végeztem (3.1 ábra).

Első lépésként készítettem egy 1 gigabájt méretű fájlt. Ezután megvizsgáltam, hogy mekkora különbség van, ha a hálózaton keresztül, vagy ha a lokális háttértárról olvassuk az adatokat.

Az adott környezetben 100Mbit/s sebességű UTP hálózat található, mely manapság a legáltalánosabb megoldás, és egy lokális hálózaton belül általában alkalmaznak.

Az első esetben egy olyan gépet használtam a teszt elvégzéséhez, ahol a lokális háttértár és a számítógép sebessége jelentősen nagyobb a hálózat sebességénél. Itt természetesen azt az eredményt kaptam, hogy a lokális háttértárról való olvasás jelentősen gyorsabb. A hálózatról való olvasás esetén elértem a hálózat által nyújtott maximális átviteli sebességet, amiből az következik, hogy a hálózati olvasásra használt teszt megfelelő. Ezt abból láthatjuk, hogy a kommunikáció nem jelent többletköltséget, tehát a hálózati másolás lehető legjobb esetét kaptuk.

A második esetben olyan számítógépet választottam, ami nem képes olyan gyorsan olvasni a lokális tárolóról, mint a hálózat által elérhető maximális sebesség. Itt is azt kaptam, hogy a hálózatról való olvasás esetén a sebesség kisebb volt, mint a lokális olvasás esetén.

Ebből az esetből következtethetünk arra, hogy mi történik, ha egy a jelenleginál gyorsabb hálózatot használunk, és a jelenlegi háttértárak csak annál lassabb sebességre képesek.

Ezen eredményekhez hozzá kell vennünk, hogy abban az esetben, ha egy központi SE-et több WN is igénybe vesz egy időben, akkor a hálózatról való közvetlen olvasás még ennél is lassabb lehet a terhelés miatt. Ez mellett a hálózati topológia miatt is jelentősen visszaeshet a hálózat sávszélessége. Ezzel ellentétben, ha az adatok a háttértáron vannak, akkor nem kell számolnunk a külvilág lassító tényezőivel.

Természetesen ezek a mérések nagyban függenek a adott hardvertől és hálózati kapcsolattól.

Abban az esetben, ha nem áll rendelkezésre a kívánt adat a lokális tárolón, a hálózaton keresztül elérhetjük. Ebben az esetben sem jelent a megoldás jelentős veszteséget, mivel ugyan azzal a sebességgel számolhatunk, mint ha kizárólag a hálózatról dolgoznánk. Az egyetlen veszteség abból adódhat, hogy az adat lokális jelenlétének vizsgálatát el kell végezni, de ez egy jó implementáció esetén elhanyagolható.

Ha a javasolt réteget beépítjük a fájlelérés két szereplője közé, akkor számolnunk kell a komponens működése által jelentkező többletköltséggel. Ez a többletkommunikációból és a komponens működéséhez szükséges erőforrásigényből adódik. Az implementáció esetén arra kell törekedni, hogy ezeket a költségeket a minimálisra szorítsuk.

3.3. Dinamikus fájleléréssel történő optimalizálás elemzése

Az adatok dinamikus elérése jelenleg csak a Site-ok szintjén van megoldva. Tehát ha dinamikusan szeretnénk olvasni egy fájlt, akkor azt a Site lokális hálózatán keresztül az ott elhelyezett SE-ről tehetjük meg. Abban az esetben, ha a dinamikus elérést a WN szintjén is megvalósítjuk, akkor például az előző fejezetben említett optimalizálási lehetőségek is kivitelezhetőek lesznek. Ezt kivitelezhetjük több módon is.

3.3.1. Megoldás szálak segítségével

Ehhez a megoldáshoz módosítanunk kell az alkalmazásokat úgy, hogy létrejöjjön egy másik szál is, amelyen keresztül az alkalmazás a fájlokat dinamikusan érheti el.

A megoldás előnyei

- Nincs szükség új komponensek telepítésére a WN-okon.

A megoldás hátrányai

- Nem biztosít optimalizálási lehetőséget a többszöri futtatás, vagy bizonyos fájlok többszöri használata esetére, mivel az ideiglenes fájlok elvesznek.
- Szükség van az alkalmazások módosítására és újrafordítására.
- Több programozási nyelven is meg kell valósítani, hogy széles körben alkalmazható legyen.

3.3.2. Új a cache-elést végző komponens készítése

Ebben a megoldásban készítenünk kell egy komponenst, mely a SE-eken tárolt fájlokat éri el. Szerver-alkalmazásként vagy szolgáltatásként fut a WN-okon és rajta keresztül folyik a távoli fájlok elérése. A kért adatokat egy cache tárolóban helyezi el. Vagy egy API-n keresztül érhetik el az alkalmazások, vagy lehetőség van egy virtuális fájlrendszer készítésére, mely a Linux VFS-én keresztül lehetővé teszi a fájlok a hagyományos fájlokhoz hasonló használatát.

A megoldás előnyei

- Tetszőleges optimalizálási stratégiát alkalmazhatunk.
- Lehetőséget biztosíthatunk az adatok asszinkron elérésére, mellyel az alkalmazások saját maguk optimalizálhatják a futásukat.
- A VFS használatával a hagyományos fájlokhoz hasonlóan használhatjuk a fájlokat, és ezáltal a fájlkezelés szempontjából tetszőleges Linux alkalmazást futtathatunk a GRID-ben.

A megoldás hátrányai

- Minden WN-ra fel kell telepíteni az új komponenst.
- Ha az API-n keresztül szeretnénk használni a fájlelérést, akkor módosítani kell a programokat.

3.3.3. Megoldás ágensek segítségével

Megoldhatjuk a dinamikus elérést úgy is, hogy egy ágenst használunk a fájlok elérésére. Ebben az esetben létrehozunk egy ágenst a WN-on, majd a job az ágensen keresztül használhatja a SE-eken elhelyezett fájlokat. Ez a megoldás az előzőnek egy a multi-ágens rendszerekbeli megfelelője.

A megoldás előnyei

- Tetszőleges optimalizálási stratégiát alkalmazhatunk.
- Lehetőséget biztosíthatunk az adatok aszinkron elérésére.
- Az ágensek tulajdonságaiból adódóan nincs szükség komponensek telepítésére, hanem tetszőleges WN-ra elküldhetjük és futtathatjuk.

A megoldás hátrányai

- A programoknak az ágenssel kell kommunikálni, és ennek megfelően módosítanunk kell a már elkészített alkalmazásokat.

3.3.4. Összehasonlítás

A fentiekből következik, hogy a két utóbbi megoldás lenne alkalmazható éles körülmények között. A jelenlegi rendszerekben, az ágensek kielégítő támogatása nélkül, a második megoldás a legalkalmasabb. Mivel azonban ehhez a megoldáshoz új komponensek telepítésére lenne szükség, nehezen integrálható a jelenlegi rendszerekbe.

Az ágensekkel való megoldás esetében meg kell teremteni a hátteret az ágensek használatához, de mivel tetszőleges ágenst használhatunk a rendszer könnyedén módosítható.

3.4. Ágensekkel történő optimalizálás elemzése

Az előző részben már előkerült a ágensek használatának lehetősége. Újabb ágens bevezetésével megoldhatjuk az adatok előreolvasását, ahogy azt korábban említettem. Az ágens a job előtt érkezik a WN-ra és elkezdi az adatok letöltését.

A jelenlegi rendszerekben az ágenst megvalósíthatjuk egy jobbal [13], melyet beküldünk a megadott WN-ra, és a végrehajtás során letölti a megfelelő adatokat.

Ebben az esetben az ütemezési döntést nem bízhatjuk a brókerre, mivel a job ütemezése előtt tudnunk kell, hogy melyik WN-on fog végrehajtódni. Mivel nem minden rendszerben van lehetőség a job számára alkalmas WN-ok listázására, ezért sajnos ebben az esetben legtöbbször nekünk kell kiválasztani a futtatás helyét.

Az is problémát okoz, hogy a jelenlegi rendszerek két szintre vannak osztva. A felső szinten legtöbbször csak a CE-eket látjuk. A WN-ok azonban a CE-ek alatt találhatók és nem feltétlenül érhetjük el őket a Site-on kívülről.

Ha a dinamikus adatelérést és az ágensekkel történő előreolvasást szeretnénk együtt használni, akkor meg kell vizsgálni, hogy milyen ágensekre van szükség, és a különböző lehetőségek milyen előnyökkel és hátrányokkal járnak.

3.4.1. Megoldás egy ágens használatával

Abban az esetben, ha minden job saját (dinamikus elérést biztosító) ágenst használ, akkor mind a két feladatot megoldhatja ez az ágens. Szükség van egy olyan ágensre, melyet a job igényeinek megfelelő futtatásra lehet felparaméterezni.

3.4.2. Megoldás két ágens használatával

Ha a dinamikus elérést biztosító ágens nem a jobhoz, hanem a WN-hoz tartozik, akkor két különböző ágenst kell alkalmaznunk. Az egyik maga az elérést biztosító ágens, míg a másik egy a jobhoz tartozó ágens. Ennek az ágensnek ismernie kell a job igényeit és a másik ágensnél igényelnie kell a szükséges adatokat. Az elérést végző ágenst fel kell készíteni arra, hogy aszinkron kérést is használhassunk. Ezen azt értem, hogy kérhetjük, hogy töltse le a megfelelő adatokat, de valójában csak később lesz rájuk szükségünk. Ennek segítségével a jobhoz tartozó ágens ezen lehetőségeket kihasználva kérheti, hogy a megfelelő adatok kerüljenek letöltésre, majd ezután, ha a job szeretné használni öket, akkor a hagyományos módon férhet hozzájuk.

3.4.3. Elemzés és összehasonlítás

A megoldások előnyei

- A jobnak nem kell tudnia arról, hogy valaki gondoskodik az adatok előreolvasásáról.

- A dinamikus elérést biztosító ágens, a fájlelérést biztosító, interfésze változatlan marad.
- Jelentősen optimalizálhatjuk a fájlok eléréséhez szükséges időt.

A megoldások hátránya

- Meg kell oldani, hogy minden jobhoz könnyedén felparaméterezhessünk egy ágenst, mely tudja, hogy a jobnak mely adatokra van szüksége.

A két ágenst használó megoldás előnyei

- Nincs szükség több fájlelérést biztosító ágensre egy WN-on, és így a több job által használt fájlok elérése tovább optimalizálható a cacheeléssel.
- A dinamikus elérést biztosító ágens könnyedén hozzáadható a WN ágenseket kezelő middleware-jéhez.
- A második esetben a job igényeit végző ágens megvalósítása nagyon egyszerűen kivitelezhető, és a dinamikus elérést végző ágenst egy jól meghatározott intefészen érheti el.
- Ezeknek köszönhetően a rendszer jobban skálázhatóvá és könnyen módosíthatóvá válik.

3.5. A workflow esetében felmerülő lehetőségek elemzése

A fent tárgyalt lehetőségek igazi előnyeit a workflow alkalmazásoknál használhatjuk ki, mivel a workflow esetében előre tudjuk, hogy milyen részjobok futtatására kerül sor és azoknak milyen adatokra van szüksége.

Ha azt a döntést, hogy melyik WN-on futtassunk egy részjobot, előbb meg tudjuk hozni mint, hogy ütemezésre kerülne, akkor a megfelelő előreolvasást végző ágens elhelyezésével az adatok letöltése jóval korábban elvégezhető.

3.5.1. Ideiglenes fájlok kezelésének optimlizálása

Abban az esetban, ha olyan adatokra van szükség, melyek az őt megelőző részjob által előállított ideiglenes fájlok, akkor újabb ágensek bevezetésével jelentős erőforrás megtakarítást érhetünk el. Meg kell oldani, hogy az előállított adatok a rákövetkező részjobok futtatási helyére kerüljenek.

A jelenlegi rendszerekben ebben az esetben a részjob befejezésekor történik a kiszámított adatok másolása, általában egy előre kiválasztott SE-re.

Megoldási lehetőségek

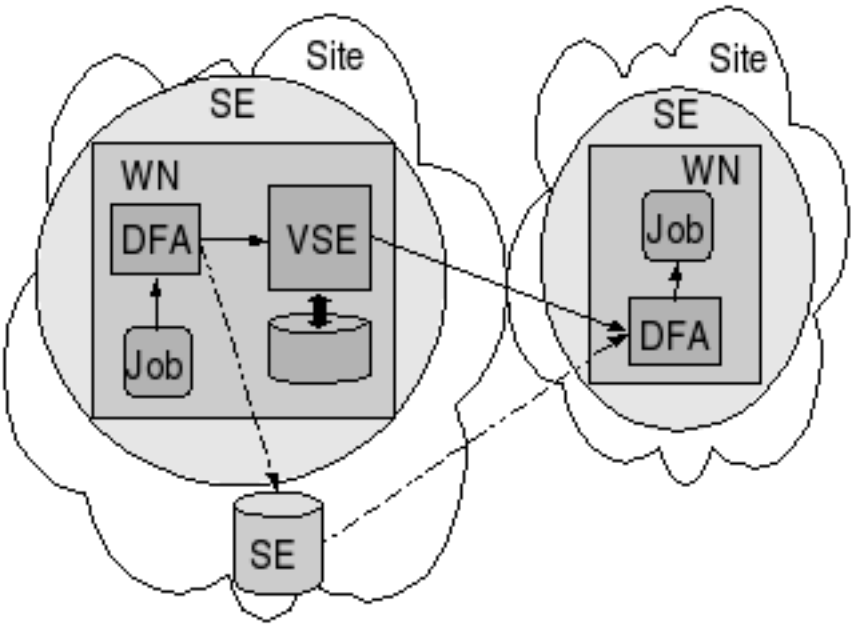
Ha a dinamikus elérést végző ágenssel(DFA) szeretnénk megoldani az ideiglenes fájloknak a megfelő helyre való juttatását, akkor szükség van arra, hogy az ilyen típusú ágensek kommunikáljanak egymással. Ha létrehozunk egy ideiglenes fájlt, akkor azt az ágensen keresztül tesszük. Az adatok a célállomáson található ágenshez való eljuttatását a létrehozás helyén található ágensnek kell megoldani. Ez a megoldás több szempontból is problémás.

Ebben az esetben a dinamikus elérést biztosító ágens szerepe kibővítésre kerül. Ennek következtében elveszítjük az ágens eredeti szerepéből adódó egyszerűséget és olyan feladatokat bízunk rá, ami elvi szempontból nem illik a szerepköréhez. Egy egyértelmű és jól definiált interfészt kellene kibővítenünk olyan lehetőséggel, mellyeknek segítségével egyenrangú felek kommunikálnak. Vagyis egy szolgáltatás szerű komponenst, mely egyértelműen kliens-szerver kapcsolatok láncában helyezkedik el, felruháznánk olyan feladatokkal, melyek elvégzéséhez egy másik és egyben egyenrangú feleket tartalmazó hálózatban kellene tevékenykednie.

Jobb és tervezési szempontból letisztultabb megoldást kapunk, ha egy virtuális SE(VSE)-et hozunk létre. Az SE-eket egy absztrakciós lépés segítségével egy ágens mögé helyezhetjük. Ez lényegében a dinamikus elérést végző kliens jellegű ágensünk szerver oldali párja. Ha ezt a felépítést használjuk, akkor az ideiglenes fájlunkat szintén egy ilyen interfészű szerver jellegű ágens kezelheti, melyet a rákövetkezők változatlanul a kliens jellegű ágenssel érhetnek el (3.2 ábra).

Láthatjuk, hogy a részjobok közti fájltovábbítás a jelenlegi és a javasolt megoldás mellet is legalább két lépésből áll. Az első lépésben a fájl felkerül a SE-re vagy Virtuális SE-re, majd a másik job letölti azt. Természetesen most eltekintettem, a dinamikus elérést biztosító ágenssel való kommunikációból adódó lépéstől. A virtuális SE esetében látszik, hogy azzal érjük el a javulást, hogy a SE-re való felmásolást lényegében a WN-on belül végezzük el. Ezen felül alkalmazhatjuk azt a megoldást, hogy a rákövetkező jobok olvashatják a már kész adatokat, mielőtt teljesen elkészül a fájl. A jelenlegi rendszerekben, a hagyományos SE esetében, ezt nem használhatjuk, mivel a fájlok vagy csak olvashatók, vagy csak írhatók, valamint a fájlok minden replikája teljes és azonos.

Azt is meg kell állapítanunk, hogy ez a megoldás csak abban az esetben alkalmazható, ha a WN-on található háttértár elegendő szabad helyet biztosít a fájl tárolására.



3.2. ábra. A virtuális SE elhelyezkedése a rendszerben.

Ezen kívül gondoskodni kell arról, hogy az ideiglenes fájl és a virtuális SE megszünjön, ha minden rákövetkező lemásolta a szükséges adatokat. Erre használhatunk egy ágenst, ami a workflow alkalmazást képviseli, és felügyeli a végrehajtását és ütemezését.

A jelenlegi rendszerekben lehetséges, hogy a fájlról replika is készül, ha a másik job egy másik Site-on fut, vagy a rendszer egyáltalán nem fog másik Site-on található WN-ot választani a replika hiánya miatt.

Az eléréshez csupán a megelőző részjob kiszámítására használt WN azonosítására van szükség. Ez az adat természetesen rendelkezésünkre áll, mivel a rákövetkező jobok elküldése pillanatában, már a megelőző job elküldésre kerül, és ennek eredménye képpen biztosan ismerjük a futásának helyét.

A megvalósításakor az SE-eket képviselő ágens egy specializált változatát kell elkészíteni, mely direkt módon a WN lokális háttértárolóján tárolja az adatokat.

Alkalmazásához el kell döntenünk, hogy melyik komponensnek kell létrehozni a virtuális SE-et. Erre két kézenfekvő megoldás is adódik. Az első esetben a dinamikus elérést biztosító komponensünk észlelheti, hogy egy ideiglenes fájlt kívánunk létrehozni az ő általa használt WN-on. Ebben az esetben létrehozhatja azt, majd felveheti vele a kapcsolatot.

Ennél tervezési és logikai szempontból talán jobb megoldás, ha a jobot képviselő ágens hozza létre a virtuális SE-et és a workflow alkalmazást kép-

viselő ágens, miután megkapta a visszajelzést, hogy minden szükséges helyre letöltötték az adatokat, jelezhet a jobot képviselő ágensnek, hogy semmisítse meg a virtuális SE-et és ő maga is fejezze be a tevékenységét. Ebben az esetben a dinamikus elérést biztosító ágensnek nem kell, hogy tudomása legyen arról, hogy nem hagyományos SE-vel kommunikál, tehát nem kell módosítanunk ezen ágens működését.

Felmerülő problémák

A virtuális SE esetében előfordulhat, hogy a CE, ahol a viruális SE működik, egy tűzfal mögött helyezkedik el. Ebben az esetben a CE-en kívülről nem biztos, hogy elérhetjük a komponenst. Ennek kiküszöbölésére találhatunk két lehetséges megoldást a 4.7. részben.

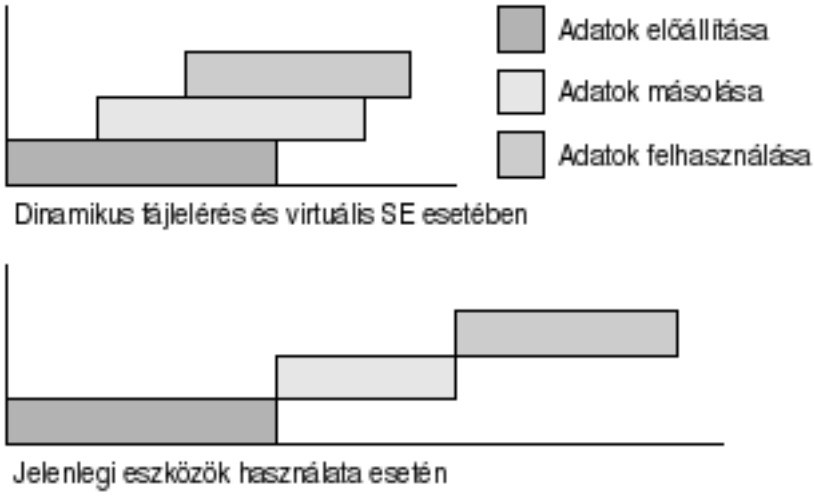
3.5.2. Optimalizálás a rákövetkezők indítási idejének hangolásával

Ha a szükséges adatok egy adott részjob esetén dinamikusan érhetők el, és az ideiglenes fájlok az adatok előállításakor dinamikusan érkeznek a WN-ra, akkor lehetőség nyílik arra, hogy a részjob indítását előrehozzuk. A részjob végrehajtása akkor lesz a legoptimálisabb, ha abban az időpillanatban indítjuk, amikor a megfelelő mennyiségű adat rendelkezésre áll ahhoz, hogy a részjob az adatokra való várakozás nélkül fusson le.

Az optimális indítás időpontját természetesen csak becsülni lehet, de ha ezt a megoldást használjuk, akkor az erőforrások felhasználása és a job befejeződése szempontjából optimális megoldást kapunk. Ha a jobot az optimális időpont előtt indítjuk el, akkor a futás közben várakoznia kell bizonyos adatokra és ezáltal feleslegesen foglaljuk az erőforrást. Ha az optimálisnál később indítjuk el, akkor viszont a késlekedésnek megfelelő idővel később fejeződik be a futás.

Jelenleg, mivel a fájlok létrehozása és letöltése nem fedheti egymást, ezért a részjob csak akkor kezdheti el letölteni az adatokat, ha az őt megelőző már befejezte a fájl adatainak írását és elhelyezte azokat valahol (3.3 ábra).

A fentiek alapján az optimalizálás menete a következőek alapján történhet. Az ideiglenes fájl tárolására egy virtuális SE-et használunk a megelőző részjob futtatásának helyén. Kiválasztjuk a rákövetkező részjob futtatásának helyét, majd odaküldjük a jobot képviselő ágenst. Az ágens folyamatosan igényli a job számára szükséges adatokat a dinamikus elérést biztosító ágensnél, és figyeli, hogy mennyi adat áll rendelkezésre. Ha az ágens úgy ítéli meg, hogy a rendelkezésre álló adatok elegendők ahhoz, hogy a jobnak ne kelljen



3.3. ábra. A statikus és a dinamikus fájlelérés összehasonlítása.

várakozni a futás közben, akkor jelzi ezt a workflow-t képviselő ágensnek, ami kezdeményezi a részjob elindítását.

Láthatjuk, hogy nem szükséges új komponens alkalmazása, és a részjob végrehajtása csak akkor nem gyorsul, ha azt az adatot állítjuk elő utoljára, amire a rákövetkezőnek először lenne szüksége. Ezen kívül megfigyelhetjük, hogy a jobot képviselő ágensnek kell tudnia, hogy a jobnak mennyi és milyen adatra van szüksége és ezek alapján eldöntenie, hogy mikor elegendő az adat a job optimális végrehajtásához. Ez nyilvánvalóan ennek az ágensnek a feladata.

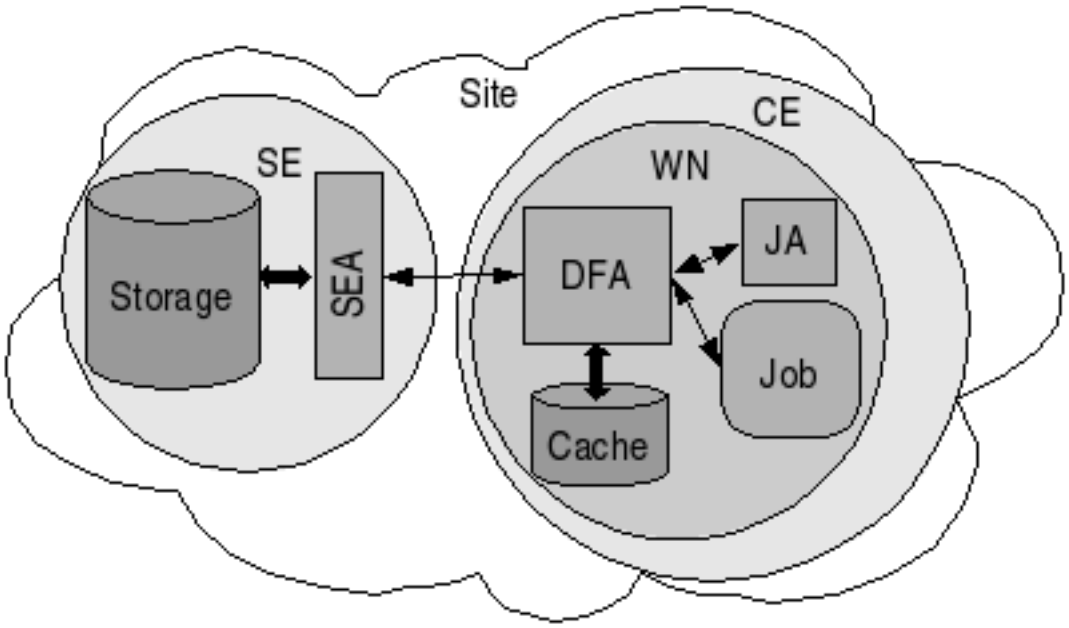
A workflow esetében ilyen lépések egymás után való végrehajtását végezzük, ezért az optimális időpont megválasztása kihatással van a rákövetkező lépésekre is.

Ennél a megoldásnál természetesen figyelembe kell vennünk, hogy a használni kívánt erőforrást a job futásának kezdete előtt is terhelnünk kell, mivel a jobot képviselő ágens tevékenykedni kezd, miközben még nem az alkalmazás birtokolja az erőforrást.

3.6. Az elemzés eredményeinek összefoglalása

A fentiekből látszik, hogy a fájlok kezelésének optimalizálásához milyen komponensekre van szükségünk (3.4 ábra).

Szükség van egy a WN-okon futó a dinamikus fájlelérést biztosító és cache-elést végző komponensre(DFA). Ez alapvető feltétele a többi lehetőség kiaknázásának. A szolgálatás-orientált környezetben ez egy szolgáltatás,



3.4. ábra. Az optimalizálásban résztvevő komponensek elhelyezkedése.

míg az ágens-alapú esetben ezt egy ágens valósíthatja meg.

A dinamikus elérést biztosító komponens szerver oldali párja egy absztrakciós szint a SE-ek elérésére(SEA). Például a jelenleg használt gLite rendszerben ezt a szolgáltatást a gLite IO komponens valósítja meg. Az ágens-alapú rendszerekben egy ennek megfelelő ágensra van szükség.

A jobot képviselő komponens(JA) egy a job igényeit ismerő alkalmazás kell, hogy legyen. A jelenlgi rendszerben egy külön jobként, vagy a jobbal együtt juttathatjuk a futtatás helyére. Az ágens-alapú rendszerekben az ágensek vándorlását biztosító tulajdonság miatt, könnyen a helyszínre juttathatjuk őket.

Ezekkel a komponensekkel elméletben biztosíthatjuk, hogy a job mindig a WN lokális háttértáráról érhesse el a szükséges adatokat. Ennek következtében a futást a lehető legoptimálisabbá tehetjük. A tevékenységek optimális időzítése esetén a fájlelérés szempontjából optimális futtatást kaphatunk.

4. fejezet

Megvalósítási javaslatok

4.1. A dinamikus elérést biztosító komponens

Ezt a komponenst egy cache-elést végző rendszerként valósíthatjuk meg, melynek egy jól meghatározott interfésze van.

A rendszer lapokra osztottan kezelheti a fájlokat a jobb adminisztrálhatóság és a nagyobb teljesítmény érdekében. Minden laphoz hozzárendeljük a legutolsó olvasás időpontját, a legutolsó olvasást végző folyamat azonosítóját valamint egy prioritást.

Szükség van az elérést biztosító interfészben olyan lehetőség bevezetésére, mellyel az alkalmazás vagy az őt képviselő komponens jelezheti, hogy egy adott fájlrészletre szüksége lesz. Ezzel aszinkron olvasást valósíthatunk meg, amivel akár a folyamat saját maga is gondoskodhat az adatok minél gyorsabb eléréséről.

A prioritás segítségével gondoskodhatunk az előre beolvasott lapok megmaradásáról azzal, hogy magasabb prioritást kapnak. Ha a job ténylegesen beolvassa, akkor a prioritás a normális szintre áll. Ha olyan adatunk van, amelyet előre beolvastunk, de a job mégsem használ fel, akkor a job befejezésekor a prioritást a normális alá csökkenthetjük. Lehetőség van a prioritások és az adat beolvasásának valószínűségének összehangolására is.

Figyelhetjük a cache állapotát, hogy blokkoljuk az előreolvasást, amíg a job nem használja fel az addig előolvasott adatok egy részét. Ezáltal elérhetjük, hogy minél később, de még időben, foglaljuk le a cache lapjait.

Meg kell vizsgálni, hogy abban az esetben, ha nem előreolvasásról van szó, érdemes-e az adatokat a cache-ben letárolni. Nagy valószínűséggel ezen adatok tárolásával nem gyorsíthatjuk a rendszert, mivel a háttértár használata miatt jelentős erőforrásigénnyel bír az olvasott lap letárolása, és mivel valószínűleg nagy mennyiségű adat olvasását végezzük a számítások során,

ezért a lap előreláthatóan nagyon rövid időn belül kikerül a cache-ből. Azt is feltételezhetjük, hogy a nagy mennyiségű adat miatt nagyon ritkán olvassák ugyanazt a fájlt ugyanazon a WN-on.

Ennek ellenére lehet olyan eset, amikor bizonyos fájlok sűrűn kerülnek letöltésre. Ebben az esetben az igényelt fájlok feljegyzésével később dönthetünk úgy, hogy egy adott fájl esetében érdemes a cache-elést elvégezni, és esetleg egy nagyobb prioritással biztosíthatjuk, hogy ne kerüljön ki a tárolóból.

Tehát az a javaslat, hogy az alapvető működéshez egy olyan komponenst kell implementálni, ami a WN-on futó joboktól és az őket képviselő ágensektől fogadja a kéréseket. Ezek lehetnek igénylések, melyek azt jelzik, hogy a későbbiekben szükségük lesz egy adott fájlrészletre, valamint lehetőség kell hogy legyen a normál, szinkron módon való olvasásra.

A fájlok létrehozását is támogatni kell. Ilyen esetben a GRID-fájlok filozófiájának megfelelően csak írható fájlokról van szó. Ekkor is alkalmazhatunk valamilyen cache-elő mechanizmust, hogy a hálózaton történő kommunikációt a lehető legoptimálisabban tudjuk elvégezni.

Az interfésznek biztosítani kell a lehetőséget arra, hogy a job, vagy az őt képviselő ágens lekérdezhesse, hogy az igényelt fájlrészletek melyike került már be a cache-be, vagy esetleg elegendő lehet, hogy hány százalékban történt ez meg. Ennek akkor van jelentősége, ha ki szeretnénk használni azt a lehetőséget, hogy a jobot az előtt indítjuk el, mielőtt a szükséges fájlok teljes előreolvasása megtörtént, vagy akár azelőtt is, hogy a megelőző részjob befejezte volna a létrehozását. A fent leírtaknak megfelelően, ekkor a jobot képviselő ágens figyeli, hogy elegendő adat áll-e rendelkezésre ahhoz, hogy a job futása optimális legyen. Ehhez azonban ismernie kell azt, hogy az igényelt adatok közül mennyi áll rendelkezésre, vagy jobb optimalizálásnál akár azt is, hogy mely részletek ezek.

4.2. Ágensek megvalósítása

Az ágens-alapú GRID esetében nagyon fontos döntés, hogy milyen eszközök segítségével valósítjuk meg az ágenseket. Több lehetséges megoldást is találhatunk, melyek eltérő lehetőségeket nyújtanak. Egy lehetőség, hogy a JADE 3.4 [22] Javában készített keretrendszert használjuk, mely a FIPA [23] által javasolt szabvány alapján készült. Használhatjuk az AgentScape middleware-t. Esetleg a szolgáltatás-alapú GRID által nyújtott lehetőségek segítségével is megvalósíthatjuk őket.

4.3. A JADE rendszer

A JADE keretrendszer a többágensű rendszerek egy szabványos megvalósítása, mely Java nyelven íródott. Az ágensek közti kommunikáció aszinkron módon történik.

A rendszer platformokra osztható. Egy platformot megfeleltethetünk a GRID rendszer egy klaszterének. A platformnak van egy main-container komponense, mely a platform magját képezi. Ezen konténer ismeretében hozhatunk létre kívűlről új ágenseket. Ez a konténer végzi a platform vezérlését és a platormok közti kommunikációt. Ehhez csatolva létrehozhatunk újabb konténereket, melyekben ágenseket helyezhetünk el.

A platformon belüli kommunikáció a Java RMI [24] protokolljának segítségével történik.

A platformok közti kommunikáció a jelenlegi rendszerben alapértelmezésben HTTP protokollon történik, valamint van lehetőség a CORBA IIOP [25] protokolljának használatára is.

A JADE előnyei

- A megvalósítás ragaszkodik a FIPA által javasolt szabványokhoz.
- Egy tesztelt és működő eszközt nyújt a több-ágensű rendszerek elkészítéséhez.
- Lehetőség van az ontológiák használatára.

A JADE hátrányai

- Az aszinkron kommunikáció miatt problémát okozhatnak a hálózati tűzfalak a platformok kötötti kommunikációban. Ennek oka, hogy a válaszolónak is el kell érnie az üzenetének célállómását, ami lehet egy tűzfal mögött, és nem lehetséges a hozzá való kapcsolódás.
- A jelenlegi verzióban nem tökéletes a viselkedések végrehajtása, mivel egy hosszabb ideig futó viselkedés problémát okozhat a rendszer más komponenseinek blokkolódása miatt.
- Az ágensek csak Java nyelven készülhetnek. Ennek következtében, mivel az ágens-alapú GRID esetében mindent ágensek valósítanak meg, ezért az egész GRID middlewareje Java nyelven kell, hogy készüljön. Ez a hatékonyság szempontjából hátrányt jelenthet.

- A platform main-containere szűk keresztmetszetet jelenthet a platformok közti kommunikáció esetén.
- A platformon kívülre való helyváltoztatást csak a rendszerhez készített kiegészítések használatával támogatja a rendszer.
- A információs rendszer skálázhatósága nincs kielégítően megoldva. A tervezéskor csak a platformon belüli felderítésre koncentráltak. Ennek következtében egy inteligens, skálázható információs rendszerre is szükség van.

4.3.1. JADE rendszer tesztelése

A JADE rendszer tesztelése érdekében készült néhány ágens a keretrendszer segítségével [13]. Ezek tesztelésére egy Condor klasztert használtunk fel. A JADE ágensek Condor[2] környezetben történő futtatásához szükség volt a megfelelő módszerek megtalálására.

Feltételek a JADE ágensek Condor klaszterben való használatához

A JADE ágensek Condor klaszterben történő használatához szükségünk van a JADE keretrendszerre és egy Condor klaszterhez való hozzáférésre. A JADE könyvtárrendszerét fel kell másolnunk arra a gépre, melyről az ágenst a kaszterbe szeretnénk beküldeni.

Main-container indítása

Első lépésben gondoskodnunk kell a JADE main-container elindításáról. A legjobb megoldás, ha a Condor központi vezérlőjén indítjuk el, mivel az ágensek beküldésénél szükség van a main-containert futtató gép címére. A main-container elindítása admininsztátori feladat. A platformon egy példányban van szükség a main-containerre, így ennek elindításáról a rendszert adminisztráló rendszergazdának kell gondoskodni.

A main-containert Condor használata nélkül is elindíthatjuk. Ehhez be kell jelentkezni a main-container futtatására kiválasztott gépre, majd a JADE dokumentációjában leírtak alapján elindíthatjuk azt [22].

Ha a Condor segítségével szeretnénk elindítani, akkor szükségünk van két fájl elkészítésére. Az első egy UNIX szkript, a második pedig a Condor jobot leíró konfigurációs fájl.

A szkriptet tartalmazó fájlt *runjadeback*-nek neveztem, ami gondoskodik egy JADE konténer háttérben való futtatásáról. Erre azért van szükség, mert azt szeretnénk, hogy a Condor job befejeződjön, miután a JADE konténer

elindul, valamint a conténert permanensen szeretnénk futtatni. A fájl csak két sort tartalmaz, azonban a tördelés miatt a *java* kezdetű sor három külön sorként jelenik meg.

```
#!/bin/sh
java -classpath
  .:jade.jar:jadeTools.jar:http.jar:iiop.jar:commons-codec-1.3.jar
  jade.Boot $1 $2 $3 $4 $5 $6 $7 $8 $9 &
```

Ezután szükség van a Condor job konfigurációs fájljára. Ezt a fájlt *jadeboot.cmd* néven mentettem el. A vanilla univerzumot kell használnunk, mivel egy szkriptet szeretnénk futtatni.

A *transfer_input_files* egy lista, ami felsorolja a JADE container elindításához szükséges fájlokat. Ebben az esetben ez csak a JADE rendszer *jar* fájljait tartalmazza. Ebből látszik, hogy a futtatást végző gépen nincs szükség a JADE rendszer telepítésére, mivel a job mellett átmásolásra kerülnek a JADE rendszer alapját képező fájlok is. Fontos, hogy a *condor_submit* parancsot a JADE könyvtárából adjuk ki, vagy a konfigurációs fájlban változtassuk meg a szükséges fájlok elérési útvonalát. A main-container indításakor nem adunk meg paramétereket a szkriptnek, mert ebben az esetben indul el a platform main-containere. A *requirements* opcióban megadhatjuk a main-container indítására kiválasztott gép címét.

```
universe        = vanilla
executable      = runjadeback
output          = jadeboot.out
error           = jadeboot.err
log             = jadeboot.log
arguments       =
transfer_input_files    = ./lib/jade.jar,./lib/jadeTools.jar,
  ./lib/iiop.jar,./lib/commons-codec/commons-codec-1.3.jar
WhenTOTransferOutput    = ON_EXIT
requirements = (machine == "nyl79")
queue
```

Ezek után a main-containert a következő paranccsal indíthatjuk el:

```
condor_submit jadeboot.cmd
```

Ellenőrizhetjük a jobot a következő paracs segítségével:

```
condor_q
```

Egy ágens elindítása

Ha elkészült az ágens és rendelkezésre állnak a Java *class* fájlok, akkor az ágenst beküldhetjük a Condor klaszterbe.

Ha az ágens osztályai egy Java csomag részei, akkor a *class* fájlokat el kell helyeznünk egy *jar* fájlban, mivel a condor nem másolja át a szükséges könyvtárszerkezetet.

Ha készítünk egy ágenst, és egy távoli helyen szeretnénk futtani, akkor a következő sorokat kell elhelyeznünk az utolsó viselkedésének *setup* nevű metódusa végen.

```
try {
     myAgent.getContainerController().kill();
} catch(StaleProxyException e) { }
```

Ennek oka, hogy az ágens futásának befejeződése esetén a konténer nem szűnik meg, amiből következik, hogy a Condor job futása soha nem fejeződik be. Ha viszont elhelyezzük ezen sorokat a megfelelő helyen, akkor csak egy ágenst tartalmazhat a konténer, mivel minden ágens befejeződne a konténer megszünésének pillanatában.

Ezek után rendelkezésünkre áll egy ágens. Szükségünk van a következő fájlokra a job futtatásához.

Az első, amit *runjade*-nek neveztem. Ez csak abban különbözik a *runjadeback*-től, hogy nem háttérben indítja el a konténert.

```
#!/bin/sh
java -classpath
  .:jade.jar:jadeTools.jar:http.jar:iiop.jar:commons-codec-1.3.jar
  jade.Boot $1 $2 $3 $4 $5 $6 $7 $8 $9
```

El kell készítenünk a job konfigurációs fájlját. A paraméterek között meg kell adnunk a JADE main-containert futtató gép címét. Meg kell adnunk a *-container* opciót, hogy egy új konténer jöjjön létre, és meg kell adnunk egy a platformon belül egyértelmű azonosítót (a példában ez a *sen*) és az ágenst tartalmazó Java *class* fájlt. A *transfer_input_files* opcióban fel kell sorolnunk a szükséges fájlokat. Ezek a JADE rendszer *jar* fájljai, valamint az ágenst megvalósító *class* vagy *jar* fájlok. A *requirement* opcióban megadhatjuk annak a gépnek a címét, ahol az ágenst szeretnénk futtatni.

```
universe        = vanilla
executable      = runjade
output          = jadesend.out
```

```
error           = jadesend.err
log             = jadesend.log
arguments       = -host nyl79 -container sen:AgentSender
transfer_input_files    = ./AgentSender.class,./
AgentSender$1.class,./lib/jade.jar,./lib/jadeTools.jar,
  ./lib/iiop.jar,./lib/commons-codec/commons-codec-1.3.jar
WhenTOTransferOutput    = ON_EXIT
requirements = (machine == "nyl78")
queue
```

Ezek után az ágenst a következő paranccsal indíthatjuk el:

```
condor_submit jadesend.cmd
```

4.3.2. JADE ágensek a jelenlegi GRID rendszerekben

Láthattuk, hogy a Condor esetében könnyedén megoldható a JADE ágensek elindítása a klaszter egy adott gépén.

Abban az esetben, ha ezt nem egy klaszterben, hanem egy GRID-ben szeretnénk megtenni, akkor azzal a problémával kerülünk szembe, hogy meg kell oldanunk, hogy egy adott WN-on induljon el a job.

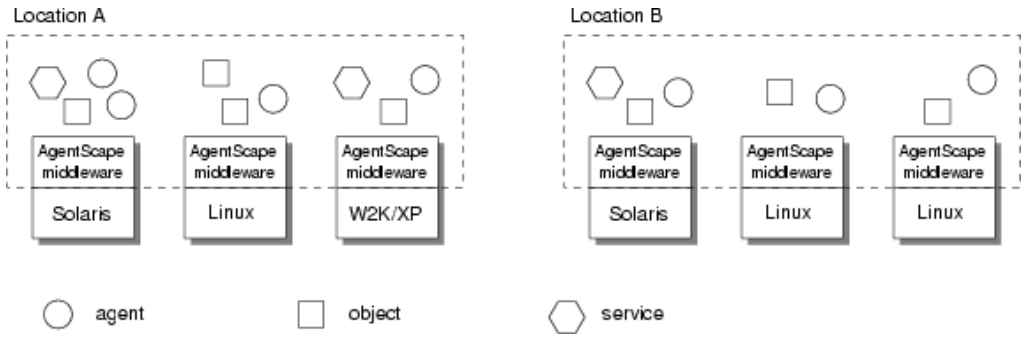
A GRID ütemezésének két szintjén kell átvinnünk a jobot. A felső esetében meg kell adnunk, hogy melyik CE-en szeretnénk futtatni, majd ezek után biztosítanunk kell, hogy a CE-en belül egy adott WN-on induljon el a job. A JDL fájlban megadhatjuk, hogy melyik CE végezze a job végrehajtását, de csak speciális esetekben adható meg a WN.

4.4. AgentScape

Az AgentScape middleware [29] tervezésekor azt a célt tűzték ki, hogy a vele készített ágens-alapú rendszer akár az internethez hasonló méreteket is elérhessen. Három szempontot vettek figyelembe a skálázhatóság vizsgálatánál: a rendszer méretét, a földrajzi távolságot az erőforrások között, valamint a felügyeletet végző domének számát.

4.4.1. A rendszer felépítése

A rendszer *location*-ökből épül fel(4.1 ábra). A rendszerben három típusú komponens fordulhat elő. Az első az ágens, mely a rendszer egy aktív egyede. A második típusú az objektum, mely passzív. A harmadik a rendszeren kívüli szolgáltatások, melyeket a rendszer ágensei használhatnak.



4.1. ábra. Az AgentScape rendszer koncepciója

Az ágensek tulajdonságai a következők:

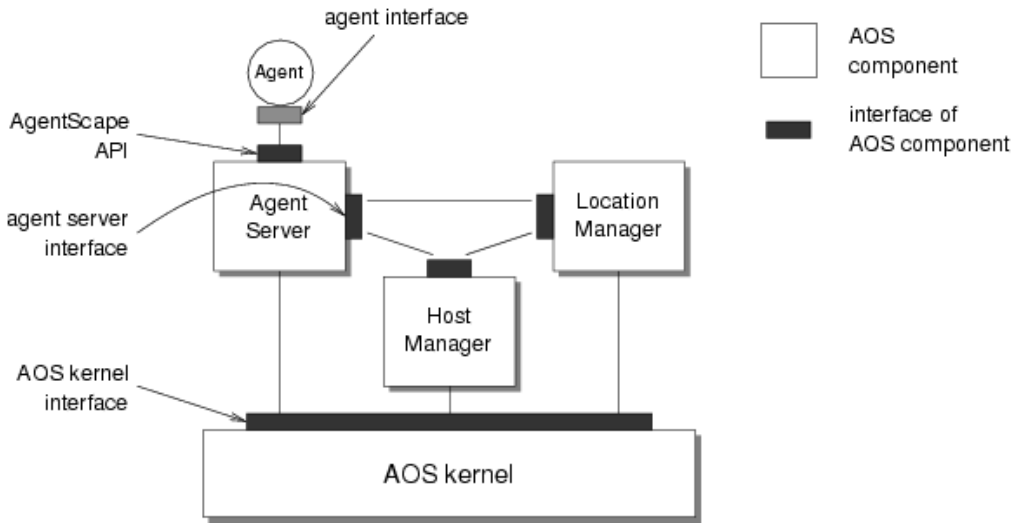
- Az ágensek maguk vezérlik a saját folyamataikat.
- Képesek kommunikálni és együttműködni.
- Képesek fogadni az információkat és válaszolni rájuk.
- Képesek kezdeményezni.

Tehát az ágensek képesek kommunikálni egymással, vagy külső szolgáltatásokkal. Képesek a helyváltoztatásra. Létrehozhatnak és megsemmisíthetnek ágenseket. Mindezen tevékenységek elvégzéséhez az ágenseknek szükségük van a megfelelő jogosultságok meglétére. Ez biztosítja a rendszer biztonságát, és védi a rendszert az illetéktelen hozzáféréstől és a rosszindulatú támadásoktól.

A külső szolgáltatások lehetnek például web-szolgáltatások, melyek elérésére a SOAP és WSDL által dinamikusan generált interfészeket használhatjuk.

Az ágensekhez, objektumokhoz és szolgáltatásokhoz hozzátartozik egy őt kiszolgáló szerver. Az ágenshez egy ágens szerver, az objektumhoz egy objektum szerver és a szolgáltatáshoz egy szolgáltatás átjáró. Ezek egy adott *location* részei, melyek biztosítják a komponensek számára a kapcsolatot a middleware-rel.

A tervezés egyik vezérelve az volt, hogy minél kisebb legyen, de elegendő lehetőséget nyújtson egy nagyméretű rendszer felépítéséhez.Ennek eredménye képpen úgy döntöttek, hogy egy többrétegű middleware-t hoznak létre(4.2 ábra). A legalsó szint a *middleware kernel*, mely az alapvető mechanizmusokat valósítja meg. A magasabb szinten találhatók a *middleware szolgáltatások*, melyek a platformfüggő lehetőségeket valósítják meg.



4.2. ábra. Az AgentScape architektúrája.

Middleware szolgáltatások

- Az ágens szerver egy ágens részére biztosítja a hozzáférést a *middleware kernel*-hez. Biztonsági okok miatt az ágenseket a hozzájuk rendelt ágensszerver elzárja a külvilágtól.
- A *location manager* irányítja a *location*-ön belül található komponenseket.
- A *location*-ön belül több *host* is lehet. Ezek vezérlését végzi a *host manager*.
- A szolgáltatás átjáró felügyelt hozzáférést biztosít egy SOAP/WSDL web-szolgáltatáshoz.

Az AgentScape előnyei

- Több programozási nyelven is készíthetünk ágenseket.
- A tervezésnél alapvető szempont volt a skálázhatóság, ami egy GRID rendszer esetében fokozott jelentőséggel bír.
- Lehetőséget biztosít a web-szolgáltatások használatára, tehát a jelenleg rendelkezésre álló GRID komponensek könnyedén felhasználhatók.
- Nagy figyelmet fordítanak a biztonságra, ami szintén elkerülhetetlen egy GRID rendszer esetén.

Az AgentScape hátrányai

- Jelenleg még sok része kidolgozás alatt áll.
- Nincs olyan szoros együttműködésben a FIPA szabványosító szervezettel, mint a JADE rendszert fejlesztők.

4.5. Ágensek megvalósítása szolgáltatások segítségével

Egy másik lehetőség az ágensek megvalósítására, ha a WSRF [26] szabványra építve web-szolgáltatásként valósítjuk meg. A jelenleg bevezetés alatt áló GRID middleware-ek erre a technikára építenek. Ha az ágenseinket web-szolgáltatásokként valósítjuk meg, akkor könnyedén alkalmazhatjuk őket a jelenelegi rendszerekben.

Ennél a megoldásnál definiálnunk kell egy interfészt, ami elegendő ahhoz, hogy az ágensek közti kommunikációt megvalósítsuk. Minimálisan ehhez elegendő egyetlen függvény, melynek meghívásával paraméterként átadhatunk egy üzenetet. Ha a kommunikáció mindkét résztvevője ágens, akkor rendelkezik ezzel a függvénnyel, és ezáltal aszinkron kommunikációt valósíthatunk meg a web-szolgáltatás által nyújtott eszközökkel.

Az interfész mögött elhelyezkedő web-szolgáltatás belső működése, már lehet olyan, hogy megfeleljen az ágensek felépítésének és működésének. Tehát lényegében a web-szolgáltatások szabványos kommunikációját használjuk az ágensek közti kommunikáció megvalósítására. A szolgáltatások nyilvántartására használt *Repository* biztosíthatja az ágensek felderítését és a velük kapcsolatos információk megszerzését.

Az ágensek helyváltoztatását is megoldhatjuk, de ennek kivitelezésekor már számos problémával találhatjuk szembe magunkat. Például problémát okozhat, hogy több programozási nyelvet is használhatunk, és nem feltétlenül tudjuk biztosítani, hogy az adott web-szolgáltatás egy adott nyelven írott változata képes futni az adott helyen.

Ez a megoldás lényegében arra használható, hogy a jelenlegi GRID rendszerekben képesek legyünk az ágensek alkalmazására anélkül, hogy módosításokat kellene eszközölnünk. Ez természetesen csak egy átmeneti megoldást jelenthet a két kölönböző elvre épített GRID között.

A web-szolgáltatások előnyei

- Szabványos kommunikációt nyújt az ágensek között.

- A web-szolgáltatások több programozási nyelven is megvalósításra kerültek, így az ágenseket is több nyelven írhatjuk.
- A web-szolgáltatások információs rendszere biztosítja az ágensek felderítését.
- A rendszer skálázhatósága megegyezik a jelenlegi GRID-rendszerekével, mivel lényegében web-szolgáltatásokat készítünk, melyek egy bonyolultabb mögöttes tartalommal bírnak.

A web-szolgáltatások hátrányai

- Szükség van egy keretrendszer megvalósítására, mely lehetővé teszi az ágensek fejlesztését, és biztosítja a velük szemben támasztott elvárások meglétét.
- Fontos a FIPA szabványok és a web-szolgáltatások eszközeinek összehangolása.
- Az ágensek helyváltoztatásának megvalósításakor nehézségekbe ütközhetünk.

4.6. Ágensek megvalósítási lehetőségeinek összehasonlítása

Az előzőekben három lehetséges megoldást is bemutattam arra, hogy egy ágens-alapú GRID rendszert építsünk fel. Természetesen egyik megoldás sem tökéletes, és mindegyik esetében szükség van bizonyos részek fejlesztésére.

Mindegyik lehetőség más szempontból jelent előnyt a másikakkal szemben. A jövőben valószínűleg több rendszer tapasztalatait felhasználva kell létrehozni egy olyan keretrendszert, ami kellően skálázható és biztonságos ahhoz, hogy GRID-et építsünk a segítségével.

A JADE rendszer legnagyobb előnye, hogy az ágensek rendszerére vonatkozó szabványok alapján készül. Sajnos a skálázhatóság tekintetében nem nyújt olyan megoldást, ami egy könnyen karbantartható rendszer építését lehetővé teszi. Azonban kiegészítésekkel elháríthatjuk ezt a problémát. A másik hátránya, hogy csak Java nyelvű megvalósítása van.

Az AgentScape rendszer legnagyobb előnye, hogy a skálázhatóság és a biztonság alapvető tervezési célkitűzés volt. Ez a rendszer nem nyújt olyan kiterjedt lehetőségeket, és a szabványokat sem követi olyan szorosan, mint

a JADE. Az előnye viszont a JADE-el szemben az, hogy több nyelven is fejleszthetünk hozzá ágenseket.

A web-szolgáltatásokkal való megvalósítás előnye csak az lehet, hogy a jelenlegi rendszerekben könnyen alkalmazhatjuk, és nincs szükség arra, hogy a jelenlegi GRID-ek middleware-jét módosítsuk. Mint már említettem, ez egy átmeneti lehetőség lehet azelőtt, hogy egy teljesen ágensekre épülő GRID middleware-t készítenénk.

4.7. A tűzfalak szerepe és problémái

A GRID rendszerekben fontos a biztonság. Ennek érdekében tűzfalakat is alkalmaznak. A tűzfal konfigurálásakor megadhatjuk, hogy mely gépek mely portjai lehessenek elérhetők a tűzfal mögött, ha a külvilág egy komponense csatlakozni kíván hozzá. Általában a kifelé irányuló kommunikáció nincs korlátozva.

Láthatjuk, hogy komoly problémát okozhat a tűzfalak használata, ha a szolgáltatás-, vagy az ágens-alapú megközelítést alkalmazzuk. Ezekben az esetekben a komponenseknek a tűzfalakon keresztül kellene kommunikálni.

Ha ehhez a kommunikációhoz csak néhány speciális portot használunk akkor azzal, hogy ezen portokat minden gépen elérhetővé tesszük, megoldható a tűzfalak problémája.

Abban az esetben, ha a használható portok nem adottak, akkor két megoldást is alkalmazhatunk.

4.7.1. Inteligens tűzfal

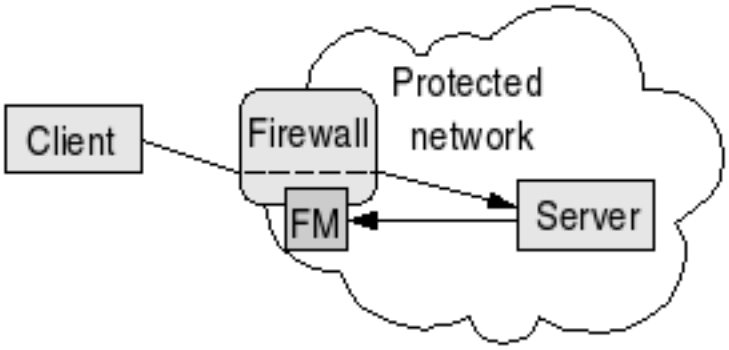
Az első, hogy a tűzfalak szabályait dinamikusan változtatjuk, ha egy olyan komponens jön létre mögötte, amely kívülről is elérhető kell, hogy legyen.

Ebben az esetben ha a rendszerhez szeretnénk igazítani a megoldást, akkor egy ágens vagy szolgáltatás reprezentálhatja a tűzfalat(FM), attól függően, hogy ágensek vagy szolgáltatások által felépített GRID rendszerről beszélünk (4.3 ábra).

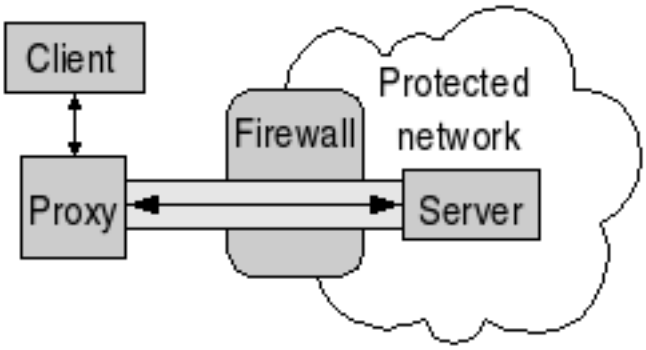
Ki kell dolgozni egy olyan interfészt, ami lehetővé teszi, hogy rajta keresztül konfigurálható legyen a tűzfal működése. Fontos, hogy figyelembe vegyük a jogosultságokat, és ezáltal megakadályozzuk a visszaéléseket, és az illetéktelenek által való manipulációkat.

Az inteligens tűzfal előnyei

- Nincs szükség új komponens alkalmazására.



4.3. ábra. Az inteligens tűzfal működése.



4.4. ábra. A proxy komponens működése.

- Nem kell különbséget tenni a komponensek között.

Az inteligens tűzfal hátránya

- Biztosítani kell, hogy a távoli komponens csak magára nyithassa ki a megfelelő portot.

4.7.2. Proxy komponens

A megoldás lényege, hogy a CE-en kívül indítunk egy proxy komponenst, mely bárhonnan elérheető és képviseli az elérhetetlen szolgáltatást vagy ágenst. Biztosítanunk kell az állandó kapcsolatot a két komponens között, mivel csak belülről építhető ki a kommunikációra használt csatorna. A proxy komponens a csatornán keresztül továbbíthatja a kéréseket a csatorna másik oldalán elhelyezkedő komponensnek (4.4 ábra). A komponensnek gondoskodnia kell a proxy komponens felszámolásáról.

Ebben a megoldásban megjelenik a komponenseknek egy másik típusa. Ez a komponens abban különbözik a megszokottól, hogy nem ő maga szolgáltatja az interfészt a kommunikációhoz, hanem egy távol elhelyezett komponens valósítja meg a lehetőségét, hogy valaki felvegye vele a kapcsolatot.

Ebben a helyzetben a legjobb megoldás, ha ezt a kettősséget a middleware elrejti, és megkíméli a komponens fejlesztőjét attól, hogy ezt megoldja. Tehát úgy kell kiegészíteni a hagyományos ágens és szolgáltatás fejlesztésére használt eszközöket, hogy a szituációtól függően képes legyen eldönteni, hogy helyben vagy távol kívánja bonyolítani a kommunikációt.

A proxy komponens előnyei

- Nincs szükség a tűzfal módosítására.
- Minden rendszerben használható.

A proxy komponens hátrányai

- A kommunikáció lassabb a két lépcső használata miatt.
- A proxy komponenst használó komponens hálózati kommunikáció segítségével kell, hogy elérje az őt képviselő proxy komponenst.
- A proxy komponenst használó és közvetlenül elérhető komponensek megvalósítása különbözhet egymástól.
- A hálózati kapcsolat kiépítése csak belülről kezdeményezhető a proxy komponens irányába, ezért folyamatosan fent kell tartani a kapcsolatot, hogy a proxy irányából is lehetőség legyen a kérések továbbítása a komponens felé.

4.7.3. A lehetőségek összehasonlítása

Mindkét megoldásnak megvan a létjogosultsága. Logikai és tervezési szempontból az inteligens tűzfal egy jobban a rendszerbe illeszthető megoldás, de ebben az esetben egy adott GRID rendszer esetén le kell cserélnünk, vagy legalábbis ki kell egészítenünk a tűzfalakat. Lehet, hogy olyan tűzfalakat alkalmazunk, amelyekre nem lehet feltelepíteni a megfelelő szoftvereket ahhoz, hogy szolgáltatásként vagy ágensként jelenjenek meg a rendszerben.

A proxy komponens esetében a megvalósításhoz módosítanunk kell a szolgáltatásokat és az ágenseket, hogy képesek legyenek egy távoli proxy-n kommunikálni, de cserébe egy olyan megoldást kapunk, amely bárhol használható, és magában a GRID-ben semmilyen változtatást nem kell végrehajtanunk.

Tehát ez a megoldás akkor jelenthet előnyt, ha a másik megoldás nem alkalmazható az adott rendszerben. Nagyon nehéz a GRID rendszerben módosításokat végezni, esetleg a GRID-et üzemeltetők nem engedélyezik, hogy új komponens, jelen esetben az inteligens tűzfalat, leváltson egy jól bevált és működő megoldást.

Tehát megállapíthatjuk, hogy mindkét megoldásra szükségünk lehet bizonyos esetekben. Mindig mérlegelni kell a GRID rendszer felépítését és az egyes megoldások megvalósításához szükséges módosítások következményeit az adott szituációban.

Abban az esetben, ha egy teljesen új GRID rendszert kívánunk kiépíteni, akkor célszerűbb a logikailag jobban a rendszerbe illeszthető inteligens tűzfalakat alkalmazni.

5. fejezet

Összegzés

Az adatintenzív alkalmazások végrehajtásának optimalizálása volt a munkám célja. Ennek érdekében az elemzés eredményeképpen eljutottam egy architektúrához, ami megoldási lehetőséget biztosít az adateléréssel járó költségek csökkentésére.

Azon áldozat meghozatala mellett, hogy előre ismernünk kell a futtatás helyét, biztosíthatjuk az adatok előreolvasását, és csökkenthetjük az adatokra való várakozást.

Fontos, hogy a javaslatok alkalmazhatóak a jelenlegi rendszerekben, valamint a jövőt jelentő ágens-alapú GRID esetében is.

Különös figyelmet fordítottam a workflow alkalmazásokra. Ezen alkalmazások esetén nagyon gyakoriak az ideiglenes fájlok, melyek esetén a fent leírtak segítségével megvalósíthatjuk, hogy miközben az egyik job létrehozza és feltölti, a másik job elkezdje olvasni őket. Ezáltal jelentős gyorsulást érhetünk el.

Megvizsgáltam a biztonság szempontjából nagy fontossággal bíró tűzfalak helyzetét, és két javaslatot is tettem, melyek bizonyos esetekben alkalmasak a problémák orvoslására.

A fentiek alapján ősszeállíthatunk a komponensekből egy rendszert, mely biztosítja a GRID-ben végrehajtott alkalmazások számára a szükséges fájlok elérését.

- Szükség van a SE-ek absztrakcióját végző komponensre.
- Ennek kliens jellegű párja minden hoszton megtalálható. Ez a komponens biztosítja a fájlok dinamikus elérését és az előreolvasás lehetőségét cache-elés segítségével. Mivel az ágensek esetén szükség van egy keretrendszerre, ami minden hosztra felkerül, ezért nem jelent problémát, hogy a dinamikus elérést biztosító komponens mindenhol megtalálható legyen.

- Használunk egy komponenst, ami a job érdekeit képviselve és annak igényeit ismerve előkészíti a szükséges adatokat a futtatás helyén.
- Alkalmazható a virtuális SE, ami biztosítja az ideiglenes fájlok helyben való tárolását, és az absztrakció miatt azonos módon kezelhető a hagyományos SE-ekkel.
- Említésre került egy komponens szükségessége, amely felügyeli a workflow alkalmazás futtatását, és koordinálja a részjobok végrehajtásakor felmerülő feladatokat.
- Megemlíthetjük az inteligens tűzfalat, ami egy interfészt biztosít a távolról történő adminisztrálásra.

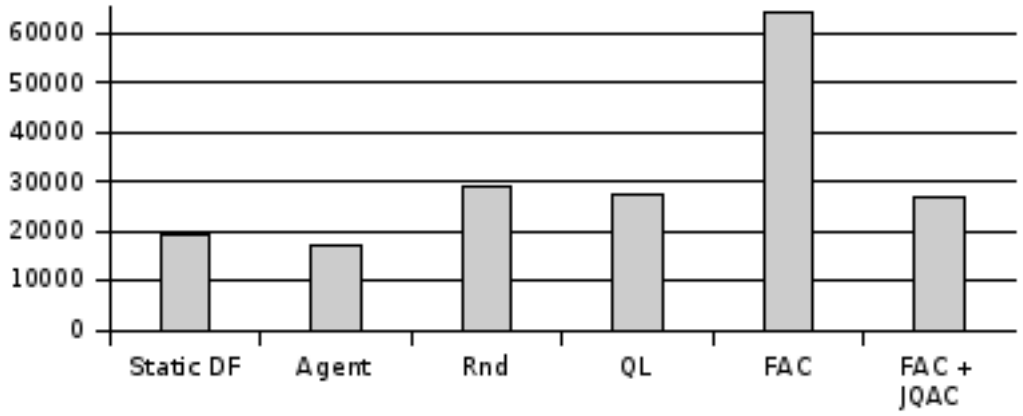
Ezen komponensek alapján összeállított rendszerben minden komponensnek jól definiált szerepe van, és az általuk megvalósított funkciók biztosíthatják a lehetőséget arra, hogy a megfelelő információk ismerete esetén az alkalmazások futtatása optimális legyen az adatok elérésének szempontjából.

A javaslatok hatékonyságát, és a bennük rejlő lehetőségeket jellemezhetjük például a [13] eredményei alapján. Ha megvizsgáljuk a cikkben szereplő stratégiákat, akkor megállapíthatjuk, hogy azok a megoldások kivitelezhetőek az általam ajánlott komponensek segítségével is. Mivel az általam javasolt megoldások nem térnek ki az ütemezési stratégiákra, ezért tetszőleges stratégia esetén kihasználhatjuk az általuk biztosított funkciókat. A cikkben leírtakon felül az általam javasolt megoldások biztosítják, hogy a fájloknak csak bizonyos részeit másoljuk le, valamint a dinamikus fájlelérés következtében továbblépést jelent, mivel nincs szükség a másolási és futtatási fázisok elkülönítésére.

Tehát a [13] cikkben végzett mérések, melyeket az OptorSim v2.0 [31] szimulációs környezet segítségével mértek, mutatja a javaslatok erejét.

Az 5.1 ábrán látható eredmények az OptorSim által mért átlagos futási ideje a joboknak a különböző stratégiák esetén. A *Static DF* a cikk által javasok statikus ütemezés. Az *Agent* a *Static DF* kiterjesztése az ágensek adta lehetőségekkel. Ezt követik a véletlenszerű(Rnd), várakozási sor hosszát vizsgáló(QL), valamint a fájlelérés költsége alapján ütemező statégiák eredményei. Az utolsó oszlop mutatja a fájleléréshez és a sorban való várakozáshoz szükséges időt.

Láthatjuk, hogy a cikk által javasolt stratégiák jelentős előrelépést jelentenek a többi stratégiához képest. Ezen eredmények tovább javíthatóak a dinamikus fájlkezelés, és a virtuális SE lehetőségeinek kihasználásával.



5.1. ábra. OptorSim szimuláció eredményeképpen kapott átlagos futási idők.

Irodalomjegyzék

[1] Foster, I.: The Grid: Blueprint for a New Computing Infrastructure, Morgan-Kaufmann, 1998, [677], ISBN-1-55860-475-8

[2] Condor Project
http://www.cs.wisc.edu/condor/ 2006. június 1.

[3] SETI@home
http://setiweb.ssl.berkeley.edu/ 2006. június 1.

[4] Foster, I. Kesselman, C. Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations, International J. Supercomputer Applications, 2001, [25]

[5] eXtensible Markup Language
http://www.w3.org/XML/ 2006. június 1.

[6] Job Description Language
http://auger.jlab.org/jdl/ 2006. június 1.

[7] Globus Project
http://www.globus.org/ 2006. június 1.

[8] Foster, I.: Globus Toolkit Version 4: Software for Service-Oriented Systems, IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, 2005, [12]

[9] Enabling Grids for E-sciencE
http://www.eu-egee.org/ 2006. június 1.

[10] LHC Computing Grid Project
http://lcg.web.cern.ch/lcg/ 2006. június 1.

[11] GridFTP
http://www.globus.org/toolkit/docs/3.2/gridftp/ 2006. június 1.

[12] László Csaba Lőrincz, Tamás Kozsik, Attila Ulbert, Zoltán Horváth: A method for job scheduling in Grid based on job execution status, Multiagent and Grid Systems, 2005, Vol. 1, [12]

[13] László Csaba Lőrincz, Attila Ulbert, Zoltán Horváth, Tamás Kozsik: Towards job scheduling in the next generation of Grid, Submitted to Grid 2006, 2006, [8]

[14] Parallel Virtual Machine
http://www.csm.ornl.gov/pvm/ 2006. június 1.

[15] Message Passing Interface
http://www-unix.mcs.anl.gov/mpi/ 2006. június 1.

[16] P-GRADE
http://www.lpds.sztaki.hu/pgrade/ 2006. június 1.

[17] gLite Project
http://glite.web.cern.ch/glite/ 2006. június 1.

[18] Relational Grid Monitoring Architecture
http://www.r-gma.org 2006. június 1.

[19] Lightweight Directory Access Protocol
http://www.ietf.org/rfc/rfc3377.txt 2006. június 1.

[20] GLUE project
http://infnforge.cnaf.infn.it/glueinfomodel/ 2006. június 1.

[21] Bellifemine, F. Poggi, A. Rimassa, G.: Developing Multi-Agent Systems with a FIPA -compliant Agent Framework, Software: Practice and Experience, 2001, Vol. 31, [26]

[22] Java Agent DEvelopment Framework
http://jade.tilab.com/ 2006. június 1.

[23] Foundation for Intelligent Physical Agents
http://www.fipa.org/ 2006. június 1.

[24] Java Remote Method Invocation
http://java.sun.com/products/jdk/rmi/ 2006. június 1.

[25] Internet Inter-ORB Protocol
http://www.omg.org/technology/documents/formal/corba_iiop.htm
2006. június 1.

[26] Web Services Resource Framework
http://www.oasis-open.org/committees/wsrf/ 2006. június 1.

[27] Simple Object Access Protocol
http://www.w3.org/TR/soap/ 2006. június 1.

[28] Web Service Definition Language
http://www.w3.org/TR/wsdl 2006. június 1.

[29] Overeinder, B.J. Brazier, F.M.T.: Scalable Middleware Environment for Agent-Based Internet Applications , In: Proceedings of the Workshop on State-of-the-Art in Scientific Computing (PARA'04), 2004, Vol. 3732, [8]

[30] Collins, J.: A Market Architecture for Multi-Agent Contracting, Proceedings of the 2nd International Conference on Autonomous Agents (Agents´98), 1998, [8], ISBM-0-89791-983-1

[31] David G. Cameron, Ruben Carvajal-Schiaffino, A. Paul Millar, Caitriana Nicholson, Kurt Stockinger, Floriano Zini: Analysis of Scheduling and Replica Optimalisation Strategies for Data Grids using OptorSim, International Journal of Grid Computing, 2004, 2(1), [13]