Grid számítások modellje, programnyelvi eszközei - Beszámoló, DemoGrid

Horváth Zoltán, ELTE, ITCS

Kivonat

A DemoGrid projekt első szakaszában egy-egy önmagában álló, mintegy 25 gépes klaszter szoftverarchitektúrájának általános vizsgálatát végeztük el. Ennek keretében megvizsgáltuk azt, hogy milyen típusú szoftvereszközökkel hogyan lehet alkalmazások készítését támogatni. A második szakaszban egy adatintenzív alkalmazást vizsgáltunk meg, illetve egy ORB(M)/PVM middleware prototípusa készült el. A harmadik szakaszban az EDG szoftver komponeseit elemeztük és javaslatot tettünk a JDL nyelv erőforrásigényt és adatelérési mintákat leíró kiterjesztésére.

1. Bevezetés

Az általános architektúravizsgálat keretében megvizsgáltuk az alábbi szofvereszközöket abból a szempontból, hogy milyen eszközökkel és hogyan támogatják Grid alkalmazások készítését.

- PVM [Fro01, Kozs01]
- MPI [Fro01, Her01]
- CORBA. Kifejlesztettünk egy speciális keretrendszert CORBA alkalmazásokhoz (ORB(M) [Ulb01, Ulb01a]) és egy Clean-CORBA interfészt ([Var01]).

A DemoGrid projekt második szakaszában folytatott kutatásainknak három fő iránya volt:

1. elosztott objektum-orientált keretrendszerek és metódushívási szemantikák [Ulb02, Ulb02a],
2. nagy mennyiségű adat feldolgozása klaszteren [HHKTU02],
3. biztonságos mobil kód készítése és alkalmazása Grid környezetben [HoKo02].

A projekt harmadik szakaszában a Grid middleware architektúrát vizsgáltuk. Az EDG Job Submission interfész, a Resource Broker és a Replica Manager együttműködését elemeztük. Javaslatot tettünk a Job Description Language (JDL) kiterjesztésére, hogy a Grid infrastruktúrát használó alkalmazásokról a middleware több információval rendelkezzen és az alkalmazás erőforrásigénye, ill. adatelérési mintái alapján határozza meg a futtatás optimális helyét és módját [HKU02].

2. Eszközök Grid alkalmazások készítéséhez

2.1. PVM

Az elosztott architektúrájú párhuzamos rendszerekben a PVM jelenti az egyik legelterjedtebb kommunikációs eszközkészletet. Használójának, a programozónak (aki általában valamilyen tudományos számítást végez az elosztott rendszer segítségével) nem nyújt olyan magasszintű szolgáltatásokat, mint mondjuk a CORBA technológia, de mindenféleképpen magasabbszintű eszköz, mint például a TCP/IP protokoll. Elterjedtsége miatt érdemes megvizsgálnunk alkalmazhatóságát Grid környezetben, és hatékonyságát, valamint kényelmességét össze kell hasonlítanunk más rendszerekével. Az ELTE Informatika Tanszékcsoport Programozási Nyelvek laboratóriumában épített klaszteren, ami 25 Linux-os PC-ből állt, üzembe helyeztünk egy PVM rendszert. Ebben a rendszerben néhány fontos párhuzamos algoritmus (asszociatív műveletek, például mátrixszorzás elvégzése; adatcsatornára visszavezethető számítások, például hatványsorok közelítése) különböző C és C++ nyelvű implementációit teszteltük [Kozs01, Bal01]. Megvizsgáltuk, hogy milyen számítás- és kommunikációigényű feladatoknál lehet a gyakorlatban is kimutatni a párhuzamos algoritmusok elméleti hatékonysági jellemzőit (például az asszociatív műveletek elvégzésénél n adatra O(n) processzoron O(log n) végrehajtási idő, vagy az adatcsatornára visszavezethető számításoknál „elég sok" adatra a megközelítőleg lineáris speedup).

2.2. MPI

Az ELTE Programozási Nyelvek laboratoriumának klaszterén installáltunk MPI függvénykönyvtárat, amelyet C nyelvű alkalmazásokkal teszteltünk [Her01].

A tesztelés során a legtöbb alapvető MPI rutin működését ellenőriztük. Az üzenetküldési technikák közül a plain és csomagolt, ill. a pont-pont és a broadcast jellegű üzenetküldést teszteltük. Megmértük a rövid (néhány byte), és a hosszú üzenetküldés (több megabyte), ill. a blokkoló és nem blokkoló változat költségeit. Ezeken kívül vizsgáltuk a standard kimenet, bemenet használhatóságát, valamint a file-kezelés lehetőségeit MPI alkalmazásokból. Az alábbi problémákat fedeztük fel: - kiderült, hogy az stdout minden processz számára elérhető, de az stdin csak a 0-s rangú processz számára. - A szeparált file i/o nem működött, talán konfigurálási okoknál fogva. - A nagyobb méretű (>30k) üzenetek küldését a rendszer nem tudta megbízhatóan kezelni (alkalmanként lefagyott).

Az MPI kisebb gondoktól eltekintve működőképesnek bizonyult.

2.3. CORBA

A Grid rendszerek egyik legfontosabb kérdése, hogy alkalmazhatók-e a meglévő objektumorientált elosztott keretrendszerek, mint például az egyes CORBA implementációk. Vizsgálataink eredményeként bebizonyosodott, hogy a CORBA alapú rendszerek csak részben képesek megfelelni a Grid által támasztot speciális követelményeknek. A hiányosságok egyik oka, a hívási szemantikák számának kötöttsége, illetve a rendszer nehéz adaptálhatósága a speciális feladatokhoz (pl. nagy mennyiségű adat mozgatása) és a számítási környezethez. A projekt keretében kifejlesztett ORB(M) rendszer, illetve mögötte álló

(PSE - Pluggable Semantic Elements) modell ezen hiányosságokat küszöböli ki. A rendszer lehetővé teszi a felhasználó számára saját, a környezetet jobban kihasználó hívási szemantikák implementálását. Méréseink szerint a speciális szemantikák (Grid, multimédia-, mobil- és csoportkommunikáció) teljesítménye esetenként kétszerese is lehet az egyik leggyorsabb CORBA implementációnak (omniORB3).

2.4. Funkcionális programozási nyelvek eszközei

2.4.1. Clean

A Concurrent Clean (lusta kiértékelésű) funkcionális programozási nyelv a következő alapvető párhuzamos programozási nyelvi elemekkel rendelkezik: annotációk, üzenetküldő és fogadó csatornák, párhuzamos folyamatok, a TCP protokollon alapuló kommunikáció elemei, valamint párhuzamos kiértékelési stratégiák. Az ELTE Programozási Nyelvek laboratoriumának klaszterén a Concurrent Clean nyelv Linux operációs rendszerre írt 1.3.3.-as fordítóját installáltuk, mely a fent említett elemek közül a TCP protokollon alapuló kommunikáció elemeit tartalmazza. A további nyelvi elemeket a jelenlegi fordító nem támogatja, így azokat implementálni kell. Ezen nyelvi elemek segítségével felépíthetőek a Grid számításokhoz szükséges, paraméterezhető, általános párhuzamos számítási sémák, a szkeletonok [ZsHo01].

Ezen túlmenően késztettünk egy Clean-CORBA interfészt is. Ily módon a Clean nyelven készített programok adatokat tudnak cserélni a CORBA szabványnak megfelelően bármely más nyelvű, CORBA interfészt használó alkalmazással is [Var01].

Haskellhez sok hasznos könyvtárat írtak, így Clean nyelvű alkalmazásokhoz ezek használhatóvá tétele jelentősen bővíti a Cleanben elkészíthető alkalmazások körét is. Ezen feladat megoldására készül egy Haskell-Clean előfordító, amely lehetővé teszi Haskell modulok Cleanből történő használatát [Heg01]. A front-end a Clean 2.0 forditóhoz készült, és képes Haskell kódot elemezni, valamint az elemzett kódból a Clean 2.0 fordító számára használható szintaxisfát generálni. Ehhez szükséges a két nyelv összehasonlítása és a különféle Haskell struktúrákkal ekvivalens Clean struktúrák megtalálása. A front-end részei: lexikografikus elemző, ami tokenizálja a Haskell kódot és alapvető konverziókat végez a tokeneken (pl. az oktális vagy hexadecimális kóddal adott karaktereket átalakítja a Clean szintaxisának megfelelő formára); szintaktikus elemző, ami elemzi a tokensorozatot és létrehozza a Clean fordító számára használható szinaxisfát; preprocesszor, ami az előző két elemzővel föl nem oldható problémák egy részét szünteti meg.

2.4.2. Haskell

A Haskell funkcionális nyelvhez több, egymástól független kiegészítést is készítettek, melyek az elosztott működést segítik. Megvizsgáltuk, hogy ezek miben különböznek egymástól, illetve a konkrét környezetünkben melyiket célszerű alkalmazni. Jelenleg a vizsgált könyvtárak közül a Distributed Haskell bizonyult a számunkra leghasználhatóbbnak. Ezt a könyvtárat, illetve a használatához szükséges Glasgow Haskell Compiler 4.08-as változatát fel is telepítettük a rendszerünkre, és ki is próbáltuk egy egyszerű kliens-szerver alkalmazással.

2.5. Elosztott objektum-orientált keretrendszerek és metódushívási szemantikák

Az elosztott objektum-orientált keretrendszerek területén folytatott korábbi kutatásaink eredményeként javasolt PSE (Pluggable Semantic Elements) model Grid-es alkalmazhatóságát vizsgáltuk a kutatás ezen szakaszában. A cél egy olyan hívási szemantika kidolgozása és megvalósítása volt, mely demonstrálja a model – és egyben a hasonló keretrendszerek – életképességét Grid alapú rendszerekben. Az új hívási szemantikát az általunk készített prototípus PSE implementációban, az ORB(M) rendszerben valósítottuk meg [Ulb02].

Az általunk javasolt új hívási szemantika a távoli metódushíváson alapuló csoportkommunikáció megvalósítására ad egyfajta módszert. Az új hívási szemantika felhasználásával lehetőség nyílik adatok elosztására kliensek egy csoportja között. Az így elosztott adatok párhuzamosan feldolgozhatók, illetve kliensenként különböző módon felhasználhatók.

A távoli metódushíváson alapuló adatszórás legnagyobb előnye más módszerekkel szemben (pl. CORBA COS Notification és Event Service), hogy tökéletesen illeszkedik az objektum-orientált programozás módszertanába, nem épít a módszertantól idegen fogalmakra. Az új hívási szemantika ORB(M) alapú implementációja teljesítményét vizsgálva megállapítottuk, hogy noha a PSE model jelentős rugalmasságot biztosít a fejlesztő számára, ez a rugalmasság mégsem párosul érzékelhető teljesítményromlással.

További, főleg elméleti eredmény az elosztott objektumorientált keretrendszerekben jól ismert csonk (stub) és váz (skeleton) fogalmak összekapcsolása és egységes kezelése. Az általunk javasolt test- (corpus) és metódus szerepek alkalmazásával vegyes (kliens és szerver) funkcionalitású objektumok hozhatók létre.

Vizsgáltuk továbbá a PSE model alacsonyszintű kommunikációs elemeit, illetve az ORB(M) alapú alkalmazások PVM-es együttműködésének lehetőségét [Ulb02a]. Az általunk javasolt szemantikus elemek (a távoli metódushívási szemantikák komponensei) lehetővé teszik a PVM, mint kommunikációs infrastruktura használatát. Az új elemeket használó hívási szemantikák segítségével PVM alapon hajthatók végre a legkülönfélébb távoli metódushívások (szinkron, aszinkron, egyirányú stb.).

A PVM-es kommunikációt megvalósító elemek bizonyos megszorítások mellett lehetővé teszik az ORB(M) alapú rendszerek tisztán PVM alapú kliensekkel történő kommunikációját. A klienseknek alkalmazkodniuk kell az ORB(M)/PVM üzenetek formátumához. A rendszer által használt üzenetformátum azonban igen egyszerű; igaz, nem megfelelő üzenetek küldése, illetve a fogadott üzenetek hibás értelmezése súlyos működési anomáliákhoz vezethet.

2.6. Nagy mennyiségű adat feldolgozása klaszteren

Az adatfeldolgozással kapcsolatos kutatásaink során egy, nagy mennyiségű adat feldolgozására alkalmas párhuzamos algoritmus tulajdonságait vizsgáltuk [HHKTU02]. Az algoritmust az általunk használt hardverkörnyezetben implementáltuk PVM és MPI keretrendszerben, valamint kliens-szerver modellt használva C és Haskell programozási nyelven. Az egyes implementációk teljesítményét megmérve különböző paraméterek esetén (feldolgozandó adat mennyisége, adathalmazok száma, stb.) képet kaptunk az absztrakt algoritmus főbb jellemzőiről, illetve az adott hardverkörnyezetnek az algoritmus működésére gyakorolt hatásáról.

Megállapítottuk, hogy az algoritmus viszonylag érzékeny bizonyos paraméterek jó megválasztására. Az implementációk egyre javuló teljesítménnyel bírtak a feldolgozandó adatok mennyiségének, illetve az adathalmazok számának növelésével. Az algoritmus kimondottan hatékonynak bizonyult valóban nagy mennyiségű adat (száz megabájtos nagyságrend) feldolgozása esetén, illetve a teljesítmény tovább javult az algoritmus bizonyos paramétereinek finomhangolásával.

2.7. Biztonságos mobil kód használata Grid környezetben

Grid számítások során szabad számítási, adattárolási kapacitást, erőforrásokat ajánlunk fel mások számára. Kérdés azonban, hogy a felajánlott erőforrásokkal az azt felhasználó fél, gondatlanságból vagy szándékosan, nem él-e vissza. Idegen kódrészletek futtatásakor a kódot fogadó és végreghajtó fél megállapodhat abban, hogy milyen tulajdonságú kódrészlet futattására kerüljön sor. A CPPCC architektúra [HoKo02] ennek a protokollnak az elemeit mutatja be. A kóddal együtt annak bizonyított tulajdonságai is eljutnak a fogadó félhez. A kód felhasználása előtt ezeket a tulajdonságokat összevethetjük a megkövetelt feltételekkel. Grid környezetben a tulajdonságok és a feltételek kifejezhetnek például erőforrásokra vonatkozó korlátozásokat is.

3. Az EDG Grid middleware vizsgálata

3.1. Az EDG szoftverarchitektúra elemzése

Megvizsgáltuk az EDG middleware szoftver architektúráját. Az EDG Resource Broker határozza meg, hogy az egyes jobok a Grid melyik klaszterén, milyen erőforrások használata mellett futtathatók. A Replica Manager feladata, hogy a job által igényelt nagyméretű adatállományokból a futtatási helyhez közeli másolatot hozzon létre. Javaslatot tettünk a Job Description Language (JDL) kiterjesztésére. A javasolt új elemek segítségével a job erőforrásigénye pontosabban leírható. A kiterjesztett JDL leírás alapján a Grid infrastruktúrát használó alkalmazásokról a middleware több információval rendelkezhet, és az alkalmazás erőforrásigénye, ill. adatelérési mintái alapján határozhatja meg a futtatás optimális helyét és módját [HKU02].

3.2. Klaszterek összekapcsolása

Az ITCS nyelvi laborban installáltuk a Globus/Condor rendszer komponenseit, hogy a DemoGrid projektben résztvevő partnerek számára a klaszter használatát lehetővé tegyük.

Hivatkozások

[HHKTU02] Horváth Z., Hernyák Z., Kozsik T., Tejfel M., Ulbert A.: A Data Intensive Computation on a Cluster. In: Distributed and Parallel Systems, Cluster and Grid Computing, Kluwer, 2002. pp. 46-53.

[Ulb01] Ulbert A.: The Architecture and Design of the ORB(M) Extensible Distributed Object Framework, October 27, 2001, 12 pages.

[Ulb01a] Ulbert A.: Performance measurements of ORB(M), October 27, 2001, 2 pages.

[Ulb02] Ulbert A: Custom Invocation semantics and Corpuses in Data Multicasting – An application of custom invocation semantics in object-oriented middleware, ECOOP 2002, PhD Workshop, Malaga, Spain, June 2002.

[Ulb02a] Ulbert A.: Semantic Elements for Parallel Computing in ORB(M). In: Distributed and Parallel Systems, Cluster and Grid Computing, Kluwer, 2002. pp. 133-140.

[HoKo02] Zoltán Horváth, Tamás Kozsik: Safe Mobile Code - CPPCC: Certified Proved-Property-Carrying Code, Workshop no.1: Resource Management for Safe Languages and Workshop no. 18: The 8th ECOOP Workshop on Mobile Object Systems: New Frontiers, 16th European Conference on Object-Oriented Programming, University of Málaga, Spain, June 10-14, 2002

[HKU02] Horváth Z., Kozsik T., Ulbert A.: Towards a better optimisation of resource allocation in Grid. Technical Report. 2002.

[Var01] Varga Z.: Developing parallel applications using Clean and the Clean-CORBA interface, 2001, 2 pages

[ZsHo01] Zsók, V. - Horváth, Z.: Skeleton Algorithms in Concurrent Clean, draft paper, 2001, 6 pages

[Fro01] Frohner Á.: Párhuzamos programozási eszközök összehasonlítása, 2001.

[Her01] Hernyák Z.: MPI tesztelése a nyelvi laboratóriumban, 2001. 11. 18., 4 oldal

[Kozs01] Kozsik T.: PVM programok írása Grid környezetben, 2001, 34 fólia.

[Bal01] Balázs G.: Párhuzamos programok implementációja PVM/cluster környezetben - mérési eredmények, 10 oldal

[Tej01] Tejfel M.: Elosztott programozás Haskell nyelven Grid környezetben, 2001, 1 oldal

[Heg01] Hegedűs H.: Transforming Haskell Structures to Clean, 2001, 6 pages