Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Juhász Sándor

Teljesítménymodellezés és futási idő előrejelzés klaszter rendszerekben

Tézisfüzet

Témavezetők:

Dr. Vajk István, Dr. Charaf Hassan
Automatizálási és Alkalmazott Informatikai Tanszék

Budapest, 2004.

Tartalom

I Bevezető 2
II Kutatási célok áttekintése 2
III A kutatás módszertana 3
IV Új eredmények összefoglalása 4
IV.1 PÁRHUZAMOS, ASZINKRON KOMMUNIKÁCIÓS CSATORNÁK MODELLEZÉSE 5
IV.1.1 Háttér *5*
IV.1.2 A tézis összefoglalása *5*
IV.1.3 Gyakorlati felhasználás *6*
IV.2 ASZINKRON SZIMMETRIKUS ÜZENETSZÓRÁS ALGORITMUS 7
IV.2.1 Háttér *7*
IV.2.2 A tézis összefoglalása *7*
IV.2.3 Gyakorlati felhasználás *9*
IV.3 SÁVSZÉLESSÉG ALAPÚ PÁRHUZAMOS TELJESÍTMÉNYMODELLEZÉS 10
IV.3.1 Háttér *10*
IV.3.2 A tézis összefoglalása *11*
IV.3.3 Gyakorlati felhasználás *13*
V Köszönetnyilvánítás 14
VI Saját publikációk jegyzéke 14
VII Irodalomjegyzék 15

I Bevezető

A szabványos kommunikációs hálózattal összekötött, általános célú személyi számítógépekből álló klaszterek a hagyományos szuperszámítógépeknél olcsóbb alternatívát kínálnak a nagy számítási igényű feladatok elvégzésére. A szabványos elemek redundáns használata miatt a hibatűrés és a skálázás megoldása lényegesen egyszerűbbé válik. A klaszterek térhódítása nemcsak a nagy kihívást jelentő tudományos problémák területén jelentős, hanem a nagyteljesítményű web szolgáltatók és adatbázis szerverek területén ipari felhasználásuk is széles körben elterjedt. Számos előnyük ellenére a klaszterek nem szorították ki teljesen az egyéb típusú, kivétel nélkül drágább számítási teljesítményt nyújtó megoldásokat (SMP, NUMA, MMP és vektor szuperszámítógépek) [23], mivel a klaszterek jelenleg két komoly hátránnyal rendelkeznek a hagyományos szuperszámítógépekkel szemben, ezek közül egyik a kommunikációs alrendszerükhöz, másik pedig programozási paradigmájukhoz kapcsolódik.

A szűkebb kommunikációs keresztmetszetből adódó hátrányok megszüntetése aktív kutatási terület, és az értekezésnek is egyik központi témája. A kommunikáció kisebb teljesítményét az okozza, hogy az általános célú kommunikációs elemek kisebb átbocsátó képességet biztosítanak drágább, speciálisan egy adott feladathoz kifejlesztett társaiknál. A teljesítmény javítására új hardver megoldások, és különféle protokoll kiterjesztések is felhasználhatók. Fontos szerephez jut az algoritmusok tervezésénél a granularitás mértékének és a kommunikációs minta a megválasztása is.

A klaszterek programozási módszerei lényegesen eltérnek a megszokott fejlesztési paradigmáktól. A szekvenciális algoritmusok elkészítéséhez képest a párhuzamosan működő feldolgozó egységek programozása már önmagában is jelentős kihívást jelent, de ennek klasszikusan kidolgozott *osztottan használt erőforrásokra épülő* módszerei a klaszterek esetében nem használhatók (a kritikus szakaszok, szemaforok, védett erőforrások, szálak, és a szinkronizált működés nem értelmezhető, vagy egész más szinten jelenik meg). A gyors közös tár erőforrás hiánya, a viszonylag nagy késleltetések és szűk sávszélességek miatt a klaszterek csak *üzenet alapú* módszerekkel, aszinkron kommunikációt alkalmazva működtethetők igazán hatékonyan. A klaszterekhez készített fejlesztési módszerek és eszközök szintén nem tökéletesek, mivel vagy a teljesítmény korlátozásával járnak (közös memória szimulációja üzenetekkel, szinkron üzenetkezelő könyvtárak), vagy csak korlátozott típusú feladatok esetében használhatók (mester-szolga megközelítés, kötegelt feldolgozás).

II Kutatási célok áttekintése

A bevezetőből kitűnik, hogy a hardver egyszerűsége ellenére a klaszterek szoftver szempontból számos, még nyitott kérdést vetnek fel. Kutatásaim során a szabványos IP alapú hálózattal összekapcsolt, hagyományos személyi számítógépekből (PC) felépülő klaszterek teljesítménymodellezésével és optimalizálásával foglalkoztam.

A különféle algoritmusok teljesítményét alapvetően befolyásolja a feladatok csomópontok között történő szétosztása, valamint az eredmény elő- és összeállításakor alkalmazott kommunikációs minta. A teljesítménnyel kapcsolatos elvárások a viszonylag lassú kommunikációs hálózat használata miatt megkövetelik a számítások és a kommunikáció átlapolásának biztosítását. Az átlapolhatóság előfeltétele az aszinkron kommunikáció alkalmazása. A szabványosnak tekinthető programozási könyvtárak (PVM [31], MPI [32]) biztosítják ugyan az aszinkron üzenetkezelés lehetőségét, de ennek használata bonyolult, így meglehetősen ritkán alkalmazzák. Ezek az üzenetkezelő könyvtárak alaphelyzetben szinkron módon működnek, a taszkok közötti üzenetváltás (feladat kiadás, begyűjtés, csoport kommunikáció) automatikusan a két taszk szinkronizálásával jár.

Kutatásaim során az üzenetküldés aszinkron működésében rejlő teljesítmény tartalékok kiaknázásával és az aszinkron működés teljesítménymodellezésével foglalkoztam. Téziseim sorra bemutatják az aszinkronitás kihasználásának és formális leírásának módszereit a párhuzamos alkalmazások létrehozásának különböző szintjein: kezdve a párhuzamos, aszinkron üzenetmodellezéstől, a pont-pont üzenetekből kialakított csoportkommunikációs primitíveken át eljutva egy teljes párhuzamos alkalmazás aszinkron modelljéhez.

A modellek elméleti alapon történő kidolgozása mellett munkámnak fontos célja volt olyan módszerek és algoritmusok kidolgozása, melyek felülmúlják a klaszterekben eddig használt megoldások hatékonyságát vagy teljesítményét, ugyanakkor elég egyszerűek ahhoz, hogy a gyakorlatban is praktikusan használhatók legyenek.

Ennek megvalósításához nem csupán a meglévő üzenetkezelő könyvtár (PVM, MPI) implementációkra támaszkodtam, hanem létrehoztam egy eseményvezérelt, teljesen aszinkron működésű, párhuzamos fejlesztési és futtatási környezetet is. A vezetésem alatt kifejlesztett Piramis nevű párhuzamos futtató keretrendszer [20] képes egységes klaszter rendszerré összekapcsolni az egyetemi laboratóriumok heterogén, különféle operációs rendszereket futtató számítógépeit. A Piramis klaszter környezet megalkotásához és különféle megoldásaihoz több tudományos publikációm kapcsolódik [2][3][8][11][12]. A rendszer hasznos tesztkörnyezetül szolgált a tézisek verifikációjához, mivel az új elgondolások alátámasztására készített gyakorlati tesztek és mérő alkalmazások futtatása részben ebben a környezetben történt.

III A kutatás módszertana

A klaszterek teljesítménymodellezésének alapja a különféle teljesítménymértékek (futási idő, hatékonyság, gyorsulás, erőforrás kihasználás) zárt matematikai alakban történő felírása a különféle rendszer és alkalmazás paraméterek függvényében (csomópontok száma, teljesítménye, hálózati sávszélesség, probléma méret, elosztási algoritmus stb.). A teljesítménymértékek matematikai modellje – a végrehajtandó feladat és a futtató környezet egyes paramétereinek ismeretében – a konkrét futtatások elvégzése nélkül is képes előre jelezni, hogy lehet-e egyáltalán, és ha igen, milyen módszerrel érdemes az adott feladatot a vizsgált hardver környezetben elosztottan megoldani. Ugyanez a módszer választ ad arra a kérdésre is, hogy a különféle paraméterek változtatása (probléma méret, csomópontok száma, terhelés kiegyenlítés) hogyan befolyásolja az eredmény előállításának sebességét, az előállításához szükséges munkaráfordítást és a számítás hatásfokát.

A megfelelő modell kialakítását a vizsgált folyamat komplexitásának elvi meghatározásával kezdtem. Ez a feladat jellegétől függően történhet az elvégzett lépések száma, vagy a nem szomszédos adathozzáférések (memória vagy diszk *cache miss*) mennyisége alapján. Amennyiben az algoritmus komplexitása elvi megfontolásokkal csak nehezen meghatározható, a futási időkre általános parametrikus modellek is illeszthetők. A modellben található konstans értékek identifikációja egyszerű előzetes mérések alapján történik. A konstansok meghatározása után a modell szélső- és határértékeink meghatározása megmutatja az optimális rendszerkonfigurációt, és felhívja a figyelmet a szűk keresztmetszetekre. A modell lehetőséget ad a kidolgozott algoritmusok különféle változatainak elvi vizsgálatára.

A modell helyességét, illetve az algoritmus gyakorlati teljesítményét különféle paraméterekkel végzett mérésekkel kell bizonyítani, és megmutatni, hogy ezek használata előnyösebb az eddigi referencia eredményekhez képest. A megalkotott modell és a kidolgozott algoritmusok gyakorlati verifikációját klaszter környezetben elkészített teszt implementációk segítségével végeztem.

A programrészletek és alkalmazások különféle paraméterek melletti futására adott becslések nem csupán az optimális paraméterek kiválasztására használhatók fel, hanem terhelés elosztó és erőforrás hozzárendelő algoritmusok bemeneti adatául is szolgálhatnak.

IV Új eredmények összefoglalása

Kutatómunkám eredményét három tézisben foglaltam össze. Mivel a klaszterek teljesítménymodellezésének kritikus, alapvető részét képezi a kommunikációs modell, *első tézisem* a klaszterekben előálló speciális körülményeket építi be a hagyományos kommunikációs formulákba. A modell az aszinkron kommunikáció egy eddig nem vizsgált teljesítmény növelési lehetőségét elemzi, leírva több párhuzamos kommunikációs csatorna egyidejű működésének adatátvitelre gyakorolt hatását. Ez a módszer az üzenet méretektől és a kommunikációs alrendszer implementációjától függően akár többszörös sebességnövekedéssel is járhat, mely a megalkotott párhuzamos aszinkron kommunikációs modellem segítségével jól leírható.

A párhuzamos rendszereken történő fejlesztés megkönnyítésére szolgáló üzenetkezelő könyvtárak nemcsak az egyszerű üzenetküldés és -fogadás lebonyolítását végzik el, hanem erre építve különféle összetett csoport kommunikációs elemeket (ún. kommunikációs primitíveket) is a fejlesztők rendelkezésére bocsátanak. Az első tézisben megfogalmazott eredmények alapján, különösen kisebb méretű üzeneteknél, már önmagában is nagy jelentőséggel bír az aszinkron kommunikáció alkalmazása, de megfelelő új algoritmusok alkalmazásával ez az előny tovább növelhető. A *második tézis* egy újfajta, egy mindenkinek üzenetszórás (*broadcast*) típusú csoport-kommunikációs primitívet definiál. Az üzenetszórás primitív jelenleg használt implementációi különféle architektúrákban (lánc, hiperkocka, fa) rendre $O(p)$, $O(dp^{1/d})$, $O(\log_b p)$ komplexitású megoldásokat jelentenek, ahol a p a célcsomópontok számát, d a hiperkocka dimenzióját, és b a fa topológia csomópontonkénti elágazásainak a számát jelenti. Ezzel szemben az általam javasolt új módszer klaszter architektúrában a résztvevő csomópontok p számától gyakorlatilag független, $O(1)$ komplexitású megoldást ad.

A különféle teljesítménybecslő módszerek a feldolgozó, kommunikációs és háttértár kezelő alrendszerek modelljeiből építkeznek. Általánosan elterjedt módszer, hogy a párhuzamos algoritmust különálló számítási, kommunikációs és I/O lépések sorozataként írják le. A módszer használatánál, egyszerűsége ellenére, komoly hátrányt jelent, hogy a különféle típusú lépések átlapolódását nem kezeli. A háttérben zajló aszinkron kommunikáció leírására született különféle modellek általában meglehetősen bonyolultak, így az összetett algoritmusok modellezése nehézkessé válik. A *harmadik tézisem* egy sávszélesség alapú, átlapolódást is figyelembe vevő modellt mutat be, mely egyszerűen alkalmazható, ugyanakkor hatékony futási idő előrejelzést tesz lehetővé.

A felsorolt tézisek mindegyike a PC alapú klaszterekhez kapcsolódik, és érvényességi körük főleg erre az architektúra típusra korlátozódik. A mai klaszter rendszerekben a csomópontok közötti kommunikáció gyakorlatilag kivétel nélkül teljesen kapcsolt (*fully switched*) hálózaton zajlik, így az elméleti fejtegetések és mérések is kizárólag az ilyen virtuális crossbar rendszerre koncentrálnak. A példák és a tézisek validálásához szükséges párhuzamos algoritmusok üzenet alapú programozási paradigmát használnak, és részben MPI, részben a saját fejlesztésű Piramis klaszter környezetben kerültek kifejlesztésre és futtatásra.

A tézisek kidolgozásánál különösen ügyeltem arra, hogy az elvi eredményeket a gyakorlatba is átültessem, ezért minden egyes elméleti eredmény alátámasztására több méréssorozatot is elvégeztem. A mérések bizonyítják, hogy a tézisekben megfogalmazott eredmények jól használhatók a gyakorlatban is, és különféle minőségi paraméterekben felülmúlják az eddig alkalmazott megoldásokat.

IV.1 Párhuzamos, aszinkron kommunikációs csatornák modellezése

[6][13][14]

IV.1.1 Háttér

A klaszterek teljesítménymodellezésében kiemelt hangsúlyt kap a kommunikációs alrendszer modelljének pontossága, mivel a nagyobb – négynél több számítógépből álló – PC klaszterek általában nem tartalmaznak közös erőforrást, emellett az általános célú kommunikációs alrendszerük nagyobb késleltetéssel rendelkezik, és kisebb sávszélességet biztosít más párhuzamos architektúrákhoz képest.

A teljesítmény kritikus kommunikáció fejlesztésére számtalan próbálkozás történt és történik, kezdve a hálózati kártya meghajtók felhasználói (*user*) szintre emelésétől (U-Net), azok a *kernel* szintű újraírásán át a hálózati protokollok lecseréléséig (ATM, Gamma [29]). A technológia fejlődése nagyban hozzájárul az egyre újabb és nagyobb teljesítményű hardver megoldások elterjedéséhez (Fast/Gigabit Ethernet [30], SCI [24], Myrinet [25], Quadrics [26], or InfiniBand [27]). A fenti módszerek jelentősen hozzájárulnak a teljesítmény növeléséhez, azonban a párhuzamos rendszerek másik célját, a széleskörű alkalmazhatóságot biztosító hordozhatóságot egyáltalán nem tartják szem előtt. Ebben a tézisben az aszinkron kommunikációt vizsgálom meg részletesen, mivel ez hordozható módon biztosítja az adatátvitel teljesítményének növekedését.

Az aszinkron (nem-blokkoló jellegű) kommunikáció két jelentős előnnyel jár a szinkron (blokkoló) változathoz képest: először is lehetővé teszi a kommunikációs és a számítási lépések átlapolódását, másodsorban pedig biztosítja, hogy több üzenet is egyszerre, párhuzamosan továbbítódjon. Míg az első okot viszonylag gyakran említik a párhuzamos feldolgozásról szóló irodalomban, a második tipikusan a klaszter rendszerekben jut szerephez. Az irodalomban több helyen olvasható [28][29][30], és saját vizsgálataim [6][14] is megerősítik, hogy a kommunikációs alrendszerek teljesítményét nem annyira a hardver felépítés, hanem gyakran sokkal inkább a szoftver többletmunka (*overhead*) mennyisége határozza meg (kommunikációs minta, másolások száma a rétegek között, operációs rendszer időzítések, implementáció hatékonysága). Ennek következménye, hogy a teljes hardver sávszélesség kihasználása érdekében érdemes a kommunikációs csatornák működését átlapolni, mivel ez jelentős teljesítmény növekedéssel jár.

A klaszterekben is alkalmazható hagyományos kommunikációs modellek [31][35][36][37][38] alapvetően lineárisak, és a t_c kommunikációs idő kiszámításához az üzenet n hosszát és a t_0 alapvető késleltetési időt veszik figyelembe:

$$t_c(n) = t_0 + nt_d = t_0 + n\frac{1}{b_{eff}} \tag{1}$$

ahol t_d az egy adategység átviteléhez szükséges idő (a b_{eff} effektív sávszélesség reciproka). A t_0 és t_d konstansok származtatása tipikusan különböző hosszúságú, oda-vissza üzenetváltások idejének lemérésével történik (ún. *ping-pong benchmark*).

IV.1.2 A tézis összefoglalása

a) Megmutattam, hogy személyi számítógép alapú klaszterekben a kommunikáció teljesítménye (az effektív sávszélesség, és nem egymásra épülő üzenetek esetén az átlagos válaszidő is) hordozható módon javítható nem-blokkoló kommunikációs primitívek párhuzamos alkalmazásával. A tézis **a)** pontjának alátámasztására az általam kidolgozott, forráskód szinten hordozható Piramis rendszer kommunikációs alrendszerét használtam, mely Windows és Linux operációs rendszerek alatt is képes a rendelkezésre álló fizikai sávszélesség jobb kihasználására párhuzamos kommunikációs csatornák alkalmazásával [14]. A teljesítmény növekedés

elsősorban abból származik, hogy a szabványos (TCP/IP) kommunikációs átviteli protokoll és különösen annak hordozható (tehát nem egy konkrét rendszerre optimalizált teljesítményű) implementációi számtalan olyan overhead-et és hatékonyság csökkentő megoldást tartalmaznak, melyek a párhuzamos csatornák használatával átlapolódnak, így negatív hatásaik eltakarhatók.

b) Az ugyanazért a fizikai átviteli kapacitásért versengő párhuzamos csatornák **a)** pontban leírt működésének modellezésére a lineáris modell nem jól alkalmazható, ezért a párhuzamos csatornák számától való függőség leírására a következő formulát dolgoztam ki:

$$t_c(n,c) = t_0 + n\frac{1}{b_{eff}(c)} = t_0 + n\left(u + v\frac{1}{c}\right), \tag{2}$$

ahol jelölje n az üzenet hosszát, c a párhuzamos csatornák számát, $b_{eff}(c)$ az effektív sávszélességet az egyszerre továbbított üzenetek számának függvényében. A t_0 a csatorna minimális késleltetése, az u és v konstansok az adategységenkénti átviteli időt (az effektív sávszélesség reciprokát) modellező hiperbolikus függvény előzetes mérésekből származtatott együtthatói. A fent bemutatott formula a szoftver *overhead*-ek eltakarásának telítődés jellegét modellezi, ahol az effektív sávszélesség növekedésének elméleti határt szab a hardver közeg elvi maximális sávszélessége (b_n):

$$\frac{1}{b_n} < u + v\frac{1}{c} \tag{3}$$

A modell validálására elvégzett mérések során a modell predikciós képességének jóságát az $E_{relatív}$ átlagos abszolút relatív hibával becsültem a következő képlet szerint:

$$E_{relatív} = \frac{1}{M}\sum_{i=1}^{M}\left|\frac{t_{mért}(n_i,c_i) - t_{becsült}(n_i,c_i)}{t_{mért}(n_i,c_i)}\right| \tag{4}$$

ahol M az elvégzett mérések száma, n_i az üzenetek hossza, míg c_i a kommunikációhoz felhasznált csatornák száma a i. mérés során. A (2) formulával leírt kommunikációs modell relatív hibája a különféle összehasonlító mérések során 5-10% volt, amely a hagyományos lineáris modellhez képest 2-5-ször pontosabb becslést jelentett [6][13].

A tézis részletes kidolgozását és a hozzá kapcsolódó ellenőrző mérések eredményeit az értekezés negyedik fejezete tartalmazza.

IV.1.3 Gyakorlati felhasználás

Bár a hordozhatóság és a teljesítmény követelményeinek külön-külön történő kielégítése megoldottnak tekinthető, egyidejű teljesítésük máig komoly nehézséget jelent. Az aszinkron működés teljesítmény javító hatása a számítások és a kommunikáció átlapolásában rejlik, az aszinkron kommunikáció azonban tipikusan lassabb a szinkron változatnál, hiszen a számítógép csak a háttérben foglalkozik az adatcserével. Klaszter környezetben, ahol a kommunikációs keresztmetszet korlátozásában nagy szerepet játszik a különféle forrásokból származó szoftver overhead, és a különböző csomópontok egyidejű kommunikációja is tipikusnak tekinthető, a párhuzamos csatornák használata lehetőséget ad az aszinkronitás lehetőségeinek további kihasználására (a különféle csatornák egymással is átlapolódnak, ezáltal biztosítva a minél jobb fizikai sávszélesség kihasználást). A fenti módszer egyben elegáns megoldást jelent a hordozható kommunikációs teljesítmény problémájára is.

A párhuzamos csatornákra felírt teljesítménymodell paramétereinek identifikálása néhány egyszerű kommunikációs mérés segítségével történik. A paraméterek meghatározása után a modellem jól jellemzi a csomópontok különböző kommunikációs mintáinak időigényét, így közvetlenül használható összetett teljesítménymodellekben a kommunikációs fázisok időigényének leírására, valamint megfelelő pontosságú bemenő adatként szolgálhat az erőforrás összerendelő és feladat ütemező rendszerek számára is.

IV.2 Aszinkron szimmetrikus üzenetszórás algoritmus

[5][7][16][17]

IV.2.1 Háttér

A klaszter rendszerek csomópontjai üzenet alapon kommunikálnak. A csoportos adatfeldolgozások megkönnyítésére az üzenetkezelő alrendszerek (így például a klaszterprogramozás *de facto* szabványának számító PVM [31] és MPI [32] is) ún. kollektív üzenetkezelő primitíveket definiálnak, melyek a különféle csomópontokon keletkező adatok szétosztására, összegyűjtésére és feldolgozására szolgálnak (*multicast, broadcast, scatter, gather, reduce, stb.*). A csoport kommunikációs primitívek használata egyszerűbbé és áttekinthetővé teszi a felhasználói alkalmazások szerkezetét, ugyanakkor igényes rendszereknél feltételezhető, hogy a kollektív kommunikációs primitíveket, a célrendszer sajátosságainak figyelembe vételével, minél nagyobb teljesítményre törekedve implementálják.

Az üzenetkezelő primitívek optimális implementációja a különféle architektúrákon, a csomópontokat összekapcsoló hálózati topológiák eltérése miatt, különbözhet. Az egyik, gyakran használt kollektív kommunikációs primitív a *broadcast*, mely az egy csomópontban jelen lévő adatokat változatlan formában továbbítja a csoport összes többi tagjához. Különös jelentőségét az adja, hogy használata nemcsak önmagában gyakori, hanem más, összetett primitívek építőkövéül is szolgál (*allgather*, *alltoall*, *barrier, allreduce*). A *broadcast* primitív leggyorsabb implementációi gyűrű, hiperkocka és fa topológiák esetén rendre $O(n)$, $O(dn^{1/d})$ illetve $O(\log_b n)$ komplexitással rendelkeznek [16]. A képletekben p a célcsomópontok számát, d a hiperkocka dimenzióját, és b a fa topológia csomópontonkénti elágazásainak a számát jelenti.

Az első tézisben bemutatott aszinkron kommunikációra építve kidolgoztam egy olyan kommunikációs mintát az üzenetszórás végrehajtására, mely *switching hub* segítségével összekapcsolt klaszter rendszerben a résztvevő csomópontok számától gyakorlatilag független, $O(1)$ komplexitású implementációt biztosít. A módszer aszimptotikusan nyilvánvalóan optimális, ugyanakkor mérések segítségével megmutattam, hogy a konstans együttható elegendően kicsi ahhoz, hogy a gyakorlatban használt üzenetméretek és csomópont számok nagy részében felülmúlja az egyéb implementációk teljesítményét.

IV.2.2 A tézis összefoglalása

a) Személyi számítógép alapú, *switching hub* segítségével összekapcsolt csomópontokból álló klaszterekben a *broadcast* kollektív kommunikációs primitívnek létezik – a csomópontok száma szempontjából – egységnyi, $O(1)$ komplexitású implementációja. Az algoritmus működése a következő:

1. A forrás csomóponton adott a szétküldésre szánt n byte hosszúságú üzenet, és a fogadó csomópontok címeinek p hosszúságú listája. A forrás csomópont az üzenetet p egyenlő részre osztja, a kerekítési hibák elkerülésére a következő képlet szerint:

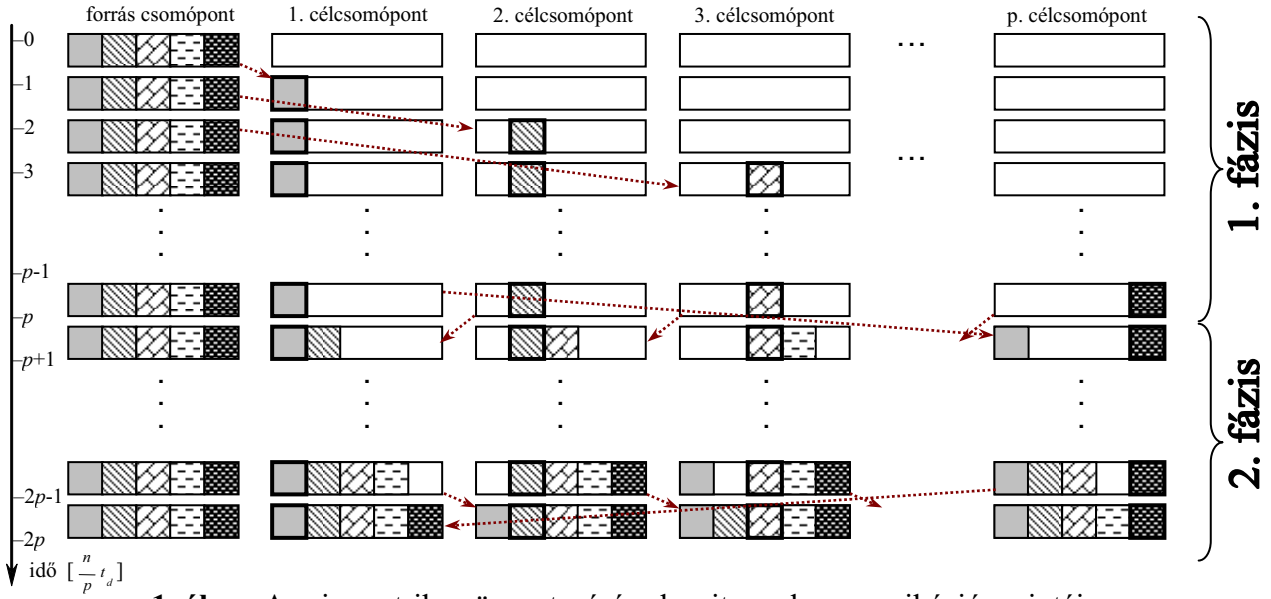
$$i.\,darab\,címe: \left\lfloor \frac{(i-1)*n}{p} \right\rfloor \qquad i.\,darab\,hossza: \left\lfloor \frac{i*n}{p} \right\rfloor - \left\lfloor \frac{(i-1)*n}{p} \right\rfloor, \tag{5}$$

   ahol i egy futóindex $[1, p]$ értelmezési tartománnyal. Az $\lfloor m \rfloor$ a lefelé kerekítés operátora, vagyis az m-hez legközelebb eső y egészet jelenti, melyre igaz, hogy $y <= m$. A forrás csomópont az $i.$ üzenetdarabot az $i.$ célcsomópontnak továbbítja, kiegészítve azt egy egyedi *id* azonosítóval, az eredeti üzenet n hosszával és a forráson kívüli összes célcsomópont címének listájával. Az i darab üzenettöredék elküldésével a forrás csomópont a ráeső feladatot elvégezte.

A fogadó csomópontok feladata az üzenetrészek fogadása, valamint a forrás csomóponttól érkező üzenetrész szétküldése az összes többi csomópontnak. Ennek érdekében a fogadó csomópontok az üzenetszórás típusú üzeneteket a következőképpen dolgozzák fel:

2. Ha egy új *id* azonosítóval rendelkező üzenet érkezik, akkor létrehoznak egy puffert az *id* azonosítóhoz tartozó üzenet darabok összeállítására, és a megérkezett darabot a helyére másolják. Az üzenetdarab helye az (5) formula alapján határozható meg, *i, p* és *n* ismeretében. A beérkező üzenet közvetlenül tartalmazza az *n*-et, és a *p* megegyezik a csomópont lista hosszával. Mivel minden üzenet küldője a kommunikációs alrendszerből lekérdezhető, ezt a címet a listában megkeresve az *i* érték is megkapható.
3. Ha a fogadó csomópont egy, már ismert *id* azonosítóval rendelkező üzenetszórás típusú üzenetet kap, akkor az *id*-hez tartozó pufferbe bemásolja az új darabot az előző pontban leírtak szerint.
4. Speciális eset, amikor az *i.* csomópont az *i.* üzenetet fogadja, mivel ez közvetlenül a forrás csomóponttól származik. Mivel a küldő csomópont nem szerepel a szétküldött csomópontlistában, ebből a fogadó csomópont tudni fogja, hogy ez az üzenet az ő darabja. Ezt a darabot úgy másolja be a helyére, hogy a csomópont listában a saját címét keresi meg, és ennek a helye alapján határozza meg a puffer címzéséhez szükséges *i* értéket. A másolás mellett a megkapott csomópontlistán is végighalad, és az összes többi csomópontnak változtatás nélkül szétküldi a kapott üzenetet. Mivel a többi csomópont számára a küldő már ez a csomópont lesz, a listabeli pozíciója alapján azok is a helyére tudják illeszteni a kapott darabot.

Az algoritmus leírása jól mutatja, hogy a darabok beérkezési sorrendje minden csomóponton tetszőleges, arra semmiféle megkötést nem kell tenni. A célcsomópontokon az utolsó üzenetdarab fogadásával és helyére illesztésével helyre áll az eredeti üzenet, és így itt is befejeződik az üzenetszórási algoritmus. Az algoritmus két fázisa (elsődleges adatkiosztás a forrás csomópontból és az első fázis csomagjainak másodlagos szétosztása) a valóságban átlapolódhat, de az 1. ábra rajzán a két fázist a jobb áttekinthetőség kedvéért külön ábrázoltam.



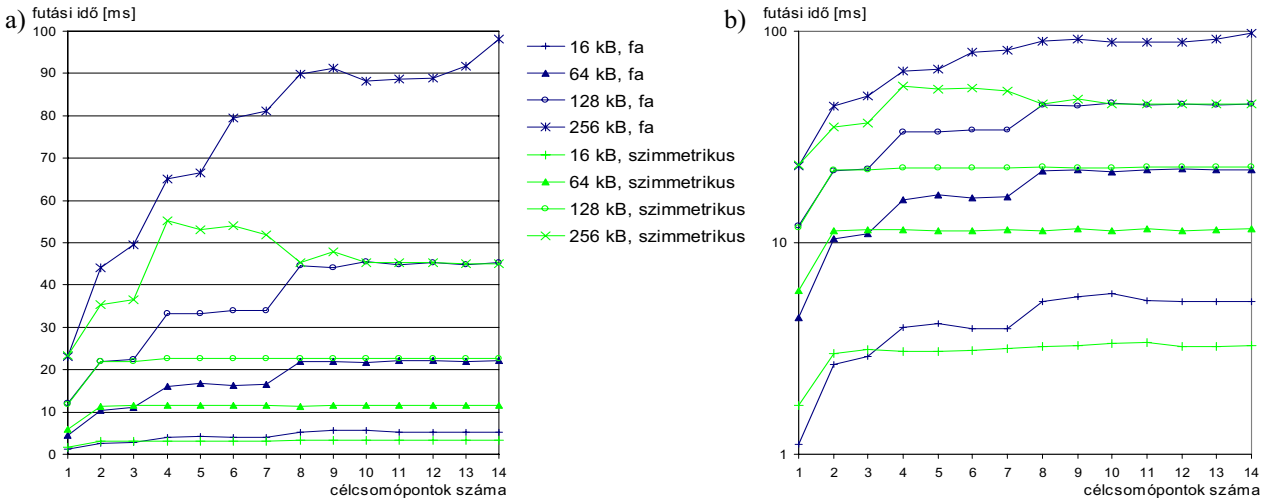
1. ábra. A szimmetrikus üzenetszórás algoritmus kommunikációs mintája a forrás csomópontban (1. fázis), és a célcsomópontokban (2. fázis)

Az algoritmus t_c futási ideje, aktív kapcsoló elemmel (*switching hub*) összekapcsolt csomópontok között a következőképpen írható fel:

$$t_c(n,p) = t_0 + nt_d + t_0 + \frac{(p-1)nt_d}{p} = 2t_0 + nt_d\left(2 - \frac{1}{p}\right) \Rightarrow O(1), \qquad (6)$$

ahol n az üzenet hossza, p a célcsomópontok száma, t_0 az adatátvitel kezdeti késleltetése, és t_d az egy adategység átviteléhez szükséges idő. A (6) képletben leírt futási idő abban az esetben érvényes, ha a forgalmat lebonyolító kapcsoló elem(ek) csomagvesztés nélkül képes(ek) az adattovábbításra. A portok telítődésének elkerülését és a forgalom egyenletességét az algoritmus szimmetriája biztosítja. A (6) formulából látható, hogy az algoritmus futási ideje a csomópontok számától aszimptotikusan független. Az elvi eredmények helyességét MPI és Piramis környezetben végzett mérésekkel egyaránt igazoltam [5][7][16][17].

b) Az egységnyi komplexitású *broadcast* kommunikációs primitív nem csupán aszimptotikusan optimális, hanem a gyakorlatban használt csomópontszámok és üzenetméretek esetén is felülmúlja a többi ismert implementációs módszert. A gyakorlati mérések alapján 2kB-nál nagyobb üzenetek esetén az **a)** pontban leírt algoritmus már gyorsabb, mint a hagyományos bináris fa alapú megvalósítás, 9 csomópont felett pedig a teljesítmény növekedés akár a 100%-ot is elérheti (az új algoritmus futási ideje fele a bináris fa alapú módszernek). A jobb összehasonlíthatóság kedvéért néhány méréssorozat eredménye a 2. ábrán látható.



2. ábra. A fa és szimmetrikus üzenetszórás teljesítményének összehasonlítása különféle üzenetméretek esetén lineáris (a) és logaritmikus (b) skálán

Az aszinkron szimmetrikus üzenetszórás részletes kidolgozását és a hozzá kapcsolódó ellenőrző mérések eredményeit az értekezés ötödik fejezete tartalmazza. Ugyanitt részletezem az algoritmus implementációjának módját és alkalmazhatóságának pontos körülményeit.

IV.2.3 Gyakorlati felhasználás

Az általam bevezetett és bemutatásra került szimmetrikus algoritmus célja, hogy kis (néhányszor tíz csomópontból álló) klaszter rendszerekben a gyakorlatban leggyakrabban használt üzenetméretek esetén [33] egy gyors, hordozható, és a csomópontok számától független megoldást biztosítson az üzenetszórásra. A módszer rendelkezik a szoftver megoldások szokásos előnyeivel, azaz rugalmas, hordozható, egyszerűen implementálható, valamint megbízható, pont-pont kommunikációra épül. Ez utóbbi biztosítja az algoritmus érzéketlenségét

a hálózati átvitel ideiglenes zavaraira. Az üzenetek darabolása és a szimmetrikus aszinkron kommunikáció teljes terhelésmegosztást biztosít, és segíti a nagy teljesítmény és a skálázhatóság elérését, mely fontos előfeltétele a gyakorlati alkalmazhatóságnak.

Az algoritmus csak tejesen kapcsolt hálózatokban nyújt optimális teljesítményt. A skálázhatóságát korlátozhatja a szűk keresztmetszetet jelentő hálózati kapcsoló, mely bizonyos forgalom fölött telítődhet. Az algoritmust működési elvéből fakadóan nem érdemes túlságosan kis méretű (<4 kB) üzenetekre alkalmazni, mivel a hálózati protokoll hatékonysága a keretméretnél jóval kisebb üzenetekre lecsökken, és a p^2 nagyságrendbe eső üzenetszám adminisztrációs terhe összemérhetővé válhat a küldés fizikai idejével.

A *broadcast* kollektív kommunikációs primitív új implementációja akár közvetlenül is használható a felhasználói alkalmazások teljesítményének javítására, de emellett beépíthető bármely aszinkron működést támogató kommunikációs alrendszerbe (üzenetkezelő könyvtárba) is. A második módszer előnye, hogy az adott alrendszerre írt, csoport kommunikációt használó felhasználói programok teljesítményét azok megváltoztatása nélkül javítja.

IV.3 Sávszélesség alapú párhuzamos teljesítménymodellezés

[1][4][9][10][15][18]

IV.3.1 Háttér

Manapság a klaszterek felépítéséhez számos különféle, nagy teljesítményű hardver és szoftver elem áll rendelkezésre, az elosztott algoritmusok optimális implementálása azonban máig számos nyitott kérdést vet fel. Ezek közül két legfontosabb az elosztott algoritmusok teljesítményének parametrikus becslése, és az elosztás optimális mértékének különféle szempontok figyelembe vételével történő meghatározása.

A fenti két probléma összefügg, hiszen ha a paraméterek (probléma méret, granularitás, kommunikációs alrendszer sebessége, csomópontok száma) függvényében sikerül meghatározni az egyes teljesítménymértékeket (futási idő, hatékonyság, gyorsulás), akkor az optimális elosztás paraméterei megkaphatók a teljesítménymérték függvényének szélsőértékeként:

$$\frac{\partial f(\mathbf{r})}{\partial \mathbf{r}_i} = 0 \longrightarrow optimális\ \mathbf{r}_i, \tag{7}$$

ahol f a teljesítménymérték alakulása az $\boldsymbol{r}$ paramétervektor függvényben. Az optimalizálni kívánt paramétert $\boldsymbol{r}_i$ jelöli.

Mivel a többi teljesítménymérték viszonylag egyszerűen származtatható a futási időből, a legtöbb modell a futási idő matematikai modelljének felépítésére koncentrál. Klaszterek modellezésénél három különböző művelet típust különböztetnek meg: számítás, kommunikáció, valamint be- és kivitel (I/O). (A háttértároló használatát nem, vagy alig igénylő, főleg számítás intenzív alkalmazásoknál a be- és kivitel kezelése kimaradhat a modellből.)

A *számítási modellek* a matematikai műveletek száma vagy a memória hozzáférési minta alapján meghatározzák az algoritmus komplexitását. Ezek után, néhány mérés alapján, regresszió analízissel kiszámítják a komplexitásban szereplő konstans értékeket. A *kommunikációs modellek* meglehetősen sokfélék, ezek közül a legfontosabbakat értekezésemben az első téziscsoporttal kapcsolatban részletesen ismertetem. Az *I/O modellek* a kommunikációs modellhez hasonlók, de lényeges egyszerűsítés, hogy a kizárólagos helyi hozzáférés miatt sem a csomópontok számát, sem a különféle összekapcsolási topológiákat nem kell figyelembe venni (kivétel: osztott közös háttértár használata).

A különféle modellek markánsan különböznek a fenti három művelettípus almodelljeinek egységes teljesítménymodellé való összeállításában. Ezek közül

legegyszerűbbek *a szinkron (blokkoló típusú műveleteket feltételező) modellek*. A klasszikus eset az elosztott algoritmust a fenti három művelettípus elkülönített, egymást követő sorozatának tekinti [35][37][39][40][41], ahol a teljes T futási idő az egyes lépések összességének idejéből áll össze:

$$T(\mathbf{r}) = \sum t_p(\mathbf{r}) + \sum t_c(\mathbf{r}) + \sum t_{io}(\mathbf{r}), \tag{8}$$

ahol t_p, t_c, t_{io} rendre a feldolgozási, a kommunikációs és az I/O elemi lépések idejét jelenti, míg az $\boldsymbol{r}$ paramétervektor az algoritmus és a futtató környezet tulajdonságait foglalja össze. A lépések idejét összegző módszer nagy előnye az egyszerűségében és a szinkron üzenetkezelő könyvtárakkal való kompatibilitásában rejlik.

A szinkron megoldások közé tartozik az Amdahl törvény [42] alkalmazása és ennek kiterjesztései is:

$$T(q) = \frac{T_{par}}{q} + T_o(q), \tag{9}$$

ahol q a csomópontok száma, T_{par} a párhuzamosítható részek teljes időigénye. A nem párhuzamosítható részekből és a kommunikációból adódó többlet futási időt a $T_o(q)$ *overhead* függvény írja le [43]. Más kutatók valószínűség alapú statisztikai és sorban állási modelleket használnak [44] a fix hosszúságú lépésekre nem jól felbontható algoritmusok teljesítmény becslésére.

Lényegesen kevesebbet foglalkozik az irodalom *az aszinkron (*a számításokkal átlapolódó kommunikáció, illetve háttértárkezelés*)* teljesítménymodellezéssel. A párhuzamos algoritmusokat ezek a modellek is az előzőekben bemutatott három alaptípushoz tartozó lépések sorozatának tekintik, itt azonban a különféle típusú lépések átlapolása megengedett. Az algoritmusok futási idő becslése ilyenkor a kommunikációs és számítási igény aszimptotikus elemzéséből származtatható [36].

A párhuzamos algoritmusok modellezésére kidolgoztam egy feldolgozási sávszélességeken alapuló módszert, mely a szinkron esetekre bevezetett megszokott almodellek használata mellett tíz százalékon belüli pontossággal képes megbecsülni az aszinkron algoritmusok futási idejét. A módszer az általam ismert egyéb aszinkron modelleknél lényegesen egyszerűbben alkalmazható, és további előnye, hogy a különféle számítási, kommunikációs és háttértár kezelési modellek tetszőlegesen kombinálhatók.

IV.3.2 A tézis összefoglalása

Def.: Adatfeldolgozás jellegű algoritmusnak tekintem az olyan, nagy mennyiségű, globális függőségi viszonyban álló adatok feldolgozására szolgáló módszereket, ahol az adatok tárolásának, továbbításának és a feldolgozásuk érdekében végzett számítási munkának az erőforrás igénye nagyságrendileg összemérhető.

a) Az azonos csomópontokból felépülő, homogén PC klasztereken futó, aszinkron (számítási, kommunikációs és I/O jellegű műveleteiket átlapolva végrehajtó), párhuzamos adatfeldolgozás jellegű algoritmusok T futási idejének becslésére a következő feldolgozási sávszélességen alapuló módszert dolgoztam ki:

1. Legyen t_c, t_{io} és t_p rendre egy N méretű probléma egy k méretű blokkjának feldolgozásához szükséges kommunikációs, I/O és számítási erőforrásidő. A b_c kommunikációs, a b_{io} I/O és a b_p feldolgozási sávszélességet rendre a k blokkméret és a feldolgozásához t_c, t_{io}, t_p idők aszimptotikus értelemben ($N \to \infty$) vett hányadosaként definiáltam:

$$b_c = \lim_{N\to\infty} \frac{N}{t_c(N,k)},\ b_{io} = \lim_{N\to\infty} \frac{N}{t_{io}(N,k)},\ b_p = \lim_{N\to\infty} \frac{N}{t_p(N,k)}. \tag{10}$$

2. Bontsuk fel a teljes párhuzamos algoritmust I darab, $s_1..s_I$-vel jelölt homogén feldolgozási szakaszra. Egy homogén szakaszon és egy csomóponton belül a kommunikációs, I/O és számítási erőforrások kihasználásának jellege nem változik. A homogén szakaszokban az adatok feldolgozása k méretű blokkokban folyik. A homogén szakaszokban az adatok feldolgozása egyszerre q darab csomóponton, k_i méretű blokkban folyik, és az $i.$ szakasz $(1 \le i \le I)$ $j.$ csomópontjának $(1 \le j \le q)$ szakaszonként a (10) szerint definiált sávszélességeit jelölje:

$$b_{c(i,j)}, b_{io(i,j)}, b_{p(i,j)} \ . \tag{11}$$

3. Egy s_i homogén szakasz T_i futási ideje legyen

$$T_i = T_{i0} + \frac{N_i}{\min_{\mathrm{j}} \min\left(b_{c(i,j)}, b_{io(i,j)}, b_{p(i,j)}\right)}, \tag{12}$$

ahol N_i az $i.$ szakaszban feldolgozott teljes problémaméret, és $T_{\mathrm{i}0}$ a különféle típusú szakaszok közötti és az állandósult állapot elérése közötti késleltetést modellezi. $T_{\mathrm{i}0}$ értéke, definíció szerint, legyen az s_{i-1} szakasz végétől az első k méretű blokk előállításának megkezdésig szükséges idő. A $T_{\mathrm{i}0}$ az összeg második tagjához képest általában elhanyagolható.

4. A teljes algoritmus T futási idejét a homogén szakaszok futási idejének összege adja:

$$T = \sum_{i=1}^{I} T_i \ , \tag{13}$$

ahol I a szakaszok száma az alkalmazásban.

A fent definiált párhuzamos, adatfeldolgozás jellegű algoritmusok optimális működésének (minimális futási idejének) eléréséhez fontos szempont a k blokkméret megfelelő megválasztása. Amennyiben az $i.$ homogén szakaszban a számítás komplexitása az $O(1)$-t meghaladja, a k feldolgozási blokkméret meghatározására a következő egyenlőtlenséggel felírható feltételrendszert dolgoztam ki:

$$\lim_{N_i\to\infty} [N_i/t_{p(i)}(N_i,k)] \ge \min\{ \lim_{N_i\to\infty} [n/t_{c(i)}(N_i,k)] \ , \lim_{N_i\to\infty} [n/t_{io(i)}(N_i,k)] \ \}, \tag{14}$$

vagyis az $i.$ szakaszban ne a $b_{p(i)}$ sávszélességű számítás késleltesse a további adatok $b_{io(i)}$ sávszélességű beolvasását és $b_{c(i)}$ sávszélességű szétosztását. Ezzel a módszerrel elérhető, hogy ne a processzor számítási teljesítménye, hanem az adatok beolvasása vagy szállítása jelentse a feldolgozás szűk keresztmetszetét.

b) A párhuzamos alkalmazások előállításának komplexitását jelentősen csökkenti a széles körben alkalmazott, mester-szolga sémára épülő szoftverfejlesztés, melynek lényege, hogy egy elosztó központ különféle, vagy különféle módon paraméterezett szekvenciálisan megoldandó feladatokat (*job*) bíz a többi csomópontra, majd összegyűjti és összeszerkeszti azok eredményeit. A módszer előnye, hogy csak a feladat elosztását végző keretrendszer igényel párhuzamos programozást, a feladat specifikus részek hagyományos, szekvenciális programfejlesztési módszerekkel is elkészíthetők. A megoldás jól kihasználja a párhuzamos végrehajtó egységek számítási kapacitását, azonban nem minden feladat bontható fel teljesen független részekre. A modell jól alkalmazható paraméter-elemzés jellegű feladatok, párhuzamos

jel- és képfeldolgozás [40], valamint más könnyen párhuzamosítható (*embarrassingly parallel*) problémák esetén (SETI@Home [45], és egyéb példák a [46]-ban).

Megállapítottam, hogy a gyakorlatban számos helyen használt mester-szolga jellegű elosztott algoritmusok klasztereken futtatva az **a)** pontban ismertetett párhuzamos adatfeldolgozás jellegű algoritmusok egyetlen homogén szakaszból álló részhalmazát képezik. Futási idejük az **a)** pontban ismertetett módszer speciális (egyszerűsített) esetének tekinthetők, ezért teljesítményük jellemzésére a következő formulát dolgoztam ki:

$$T = \frac{N}{\min\left(b_{pM}, b_{pW}, b_c, b_{io}\right)}, \tag{15}$$

ahol N a probléma méret, b_{pM} az elosztó központ (*master*) számítás-jellegű feldolgozási sávszélessége (beleértve az elosztás és az eredmény feldolgozásának számításigényét is), b_{pW} az egyes feldolgozó csomópontok (*workers*) ***együttes*** feldolgozási sávszélessége, b_c az elosztó központ effektív hálózati sávszélessége (beleértve az adatok küldését és fogadását is), és b_{io} az elosztó központ háttértárhoz kapcsolódó sávszélessége (beleérte az adatok írását és olvasását is).

A sávszélesség alapú teljesítménymodell részletes kifejtését és a modell pontosságának értékelését az értekezés hatodik fejezete tartalmazza. Ugyanitt két részletesen kidolgozott példát is szerepeltettem, egyik többlépéses összetett függőségi viszonyokat mutat be (egészek elosztott rendezése), míg a másik egy egyszerű, mester-szolga architektúrát használó megoldás (animáció elosztott sugárkövetéssel).

IV.3.3 Gyakorlati felhasználás

A téziscsoportban ismertetett megoldások egy széles probléma tartományt lefedő algoritmus osztály futási idejének becslésre adnak lehetőséget homogén klaszter rendszerekben. A sávszélesség alapú megközelítés legfontosabb újdonsága az aszinkron működés egyszerűen használható modellezésében rejlik. A módszer lehetőséget biztosít két fontos rendszer paraméter (a csomópontok száma és a feldolgozási blokkméret) optimális értékének becslésére is.

A futási idő függvényekből egyszerűen származtathatók a többi teljesítménymérték (hatékonyság, gyorsulás) függvényei is, vagyis a bemutatott módszerrel a paraméterek optimalizálása nem csupán a futási idő, hanem a többi teljesítménymérték alapján is elvégezhető. A paraméterek optimalizálása mellett az algoritmus lehetőséget biztosít a futási idő rögzített paraméterek mellett történő becslésére is, és így az erőforrás hozzárendelő és ütemező algoritmusok működésének alapjául szolgálhat.

A módszer sokoldalúságát mutatja, hogy a különféle alrendszerek modelljeire semmilyen megkötést sem tesz, így valóban képes az igen általánosan definiált adatfeldolgozás jellegű algoritmusok sokféleségéhez alkalmazkodni. Ezt a kapcsolódó publikációkban megjelenő feladattípusok sokféleségével demonstrálom (egyszerű egészek rendezése [1], adatbányászati alkalmazások [4][15][18], valamint valós idejű kép/film szintézis [20]).

V Köszönetnyilvánítás

A disszertációm elkészülte öt év kitartó munkáját tükrözi, így létrejöttében meghatározó szerepet játszott a biztos háttér, tágabb és szűkebb családom lelkesedése és biztatatása, valamint a kezdetben óriási, de az idők során egyre fogyó mértékű türelme is, melyet ezúton mindegyiküknek köszönök.

Szintén szeretnék köszönetet mondani a Ph.D. értekezés elkészítésében játszott szerepéért témavezetőimnek, Dr. Vajk Istvánnak, aki a munka szakmai színvonala felett őrködött, és Dr. Charaf Hassannak, aki a munka emberi és technikai feltételeinek megteremtésében nyújtott nélkülözhetetlen segítséget, valamint mindkettőjüknek azért, hogy a kritikus pillanatokban mindig képesek voltak megfelelő méretű és irányú lökést adni a munka folytatásához.

Fontos szerepet játszottak a dolgozat létrejöttében közvetlen kollégáim, akik hozzáállásukkal, lelkesedésükkel, konstruktív meglátásokkal vagy közvetlen szakmai és technikai segítséggel támogatták munkámat. Fontos volt számomra az is, hogy tanszékünk fiatal és lelkes csapata magas szakmai színvonalú, ugyanakkor barátságos környezetet és hangulatot teremtett.

A Piramis rendszer implementációjában játszott szerepe miatt külön köszönet illeti Csikvári András, aki több éven keresztül számtalan órát töltött a rendszer implementálásával, optimalizálásával, csinosításával és természetesen hibakereséssel.

VI Saját publikációk jegyzéke

A tézisfüzetben bemutatott eredményekhez kapcsolódó saját publikációk listája típusonként a megjelenés sorrendjében:

Folyóirat cikkek:

[1] **Sándor Juhász,** Hassan Charaf, *Performance Modelling of Cooperating Tasks in PC Clusters*, Periodica Polytechnica, Electrical Engineering, 2002 46/3-4, Budapest University of Technology and Economics, 2002, pp. 137-149.

[2] **Juhász Sándor,** *Párhuzamosság és elosztottság a klaszterek világában*, Elektrotechnika, 96. évfolyam 3. szám (2003/3), Magyar Elektrotechnikai Egyesület, Budapest, 2003, pp. 66-70.

[3] **Juhász Sándor,** *Nagy számítási teljesítményű rendszerek kialakítása személyi számítógépek összekapcsolásával*, Elektrotechnika, 96. évfolyam 4. szám (2003/4), Magyar Elektrotechnikai Egyesület, Budapest, 2003, pp. 117-121.

[4] **Sándor Juhász**, Ferenc Kovács, *A New PC Cluster Based Distributed Association Rule Mining*, Scientific Bulletin of "Politehnica" University of Timisoara, Transactions on Automatic Control and Computer Science, Vol.49 (63) 2004 No. 3, ISSN 1224-600X, Editura Politehnica, Timisoara, Romania, 2004, pp. 5-11.

[5] **Sándor Juhász**, Ferenc Kovács, *Fully Distributed Broadcast Algorithms for Small Clusters*, Scientific Bulletin of "Politehnica" University of Timisoara, Transactions on Automatic Control and Computer Science, Vol.49 (63) 2004 No. 3, ISSN 1224-600X, Editura Politehnica, Timisoara, Romania, 2004, pp. 177-183.

[6] **Sándor Juhász**, *Modeling Asynchronous Message Passing in small Cluster Environments*, International Journal on Applied Computing, ACTA Press (*submitted, under review)*

[7] **Sándor Juhász**, *A Low-Complexity Broadcast Algorithm for Fully-Switched Clusters*, Journal of Parallel and Distributed Computing, Elsevier (*submitted, under review)*

Konferencia cikkek:

[8] **Juhász Sándor**, Charaf Hassan, *Virtuális számítógép architektúra hálózati PC-kre*, Networkshop 2001 konferencia, április 18, Sopron, 2001, p. 51.

[9] **Sándor Juhász**, Hassan Charaf, *Execution Time Prediction for Parallel Data Processing Tasks*, 10th Euromicro Workshop on Parallel, Distributed and Network based Processing, IEEE Computer Society, January 9-11, Las Palmas de Gran Canaria, Spain, 2002, pp. 31-38.

[10] **Juhász Sándor**, *Elosztott algoritmus tervezés és modellezés PC klaszterekhez*, Networkshop 2002 konferencia, március 28, Eger, 2002, p. 113.

[11] **Juhász Sándor**, *Klaszter építés korszerű operációs rendszereken*, HTE-BME 2002 Diákkonferencia, május 10, Budapest, 2002, pp. 34-35.

[12] **Sándor Juhász**, Hassan Charaf, *Building Clusters on Modern Desktop Operating Systems,* 21ST IASTED International Conference on Applied Informatics (section: Parallel and Distributed Computer Networks), February 10-13, Innsbruck, Austria, 2003, pp. 547-552.

[13] **Sándor Juhász**, Ferenc Kovács, Hassan Charaf, *Asynchronous Communication Model for Cluster Systems,* 22ND IASTED International Conference on Parallel and Distributed Computer Networks, February 17-19, Innsbruck, Austria, 2004, pp. 18-23.

[14] **Sándor Juhász**, Hassan Charaf, *Exploiting Fast Ethernet Performance in Multiplatform Cluster Environment*, ACM Symposium on Applied Computing, SAC'04, March 14-17, Nicosia, Cyprus, 2004, pp. 1407-1411.

[15] **Sándor Juhász**, Renáta Iváncsy, István Vajk, *Performance Modelling of the Apriori Association Rule Mining Algorithm*, MicroCAD 2004 International Scientific Conference, University of Miskolc, March 18-19, Miskolc, 2004. pp. 205-210.

[16] **Juhász Sándor**, Csikvári András, *Csoportos üzenetszórás optimalizálása klaszter rendszerekben*, Networkshop 2004 konferencia, árpilis 5-7, Győr, 2004, p. 48.

[17] **Sándor Juhász**, Ferenc Kovács, *Asynchronous Distributed Broadcasting in Cluster Environment*, EuroPVM/MPI 2004, September 19-22, Budapest, Hungary, 2004. (*accepted for publication*)

[18] Renáta Iváncsy, **Sándor Juhász,** Ferenc Kovács, *Performance Prediction for Association Rule Mining Algorithms*, ICCC 2004, IEEE International Conference on Computational Cybernetics, August 30 - September 1, Wien, Austria, 2004, pp. 267-271.

Elektronikus publikációk:

[19] **Juhász Sándor**, *Elosztott információs rendszerek*, Elektronikus jegyzet a BME Automatizálási és Alkalmazott Informatikai Tanszékének Elosztott rendszerek című tárgyához, 1999-2000. http://avalon.aut.bme.hu/~sanyo/Elosztott_Inf_Rendszerek.zip

[20] **Sándor Juhász** et al., *The Pyramid Project*, Budapest University of Technology and Economics, Department of Automation and Applied Informatics, 2001-04. http://avalon.aut.bme.hu/~sanyo/piramis

[21] **Juhász Sándor**, *Integrált információs rendszerek*, Elektronikus jegyzet a BME Automatizálási és Alkalmazott Informatikai Tanszékének Integrált Információs rendszerek című tárgyához, 2003. http://avalon.aut.bme.hu/~sanyo/IntegraltInf.zip

[22] **Juhász Sándor**, Czuczor Szabolcs, Aszódi Barnabás: Klaszter rendszerek felépítése és programozása, Elektronikus jegyzet a BME Automatizálási és Alkalmazott Informatikai Tanszékének Szoftver Architektúrák című tárgyához, 2004. http://avalon.aut.bme.hu/education/VIAUD068/parhuzamos_rendszerek.pdf

VII Irodalomjegyzék

[23] *Top 500 Supercomputer Sites*, http://www.netlib.org/benchmark/top500.html, 2003

[24] IEEE STD 1596-1992, *IEEE standard for scalable coherent interface*, 1993.

[25] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W. Su, *Myrinet: A Gigabit-per-second Local Area Network*, IEEE Micro, 15(1):29–36,February 1995.

[26] F. Petrini,W. Feng, A. Hoisie, S. Coll, and E. Frachtenberg, *The Quadrics Network: High-Performance Clustering Technology*, IEEE Micro, 22(1), 2002. pp. 46–57.

[27] InfiniBand Trade Association, *InfiniBand Architecture Specification*, Release 1.0, October 24 2000.

[28] Martin et al., Effects of Communication Latency. Overhead and Bandwidth in a Cluster Architecture, 24th Annual International Symposium on Computer Architecture, Proceedings, Denver, 1997, pp. 85-97
[29] G. Chiola, G. Ciaccio, *Efficient Parallel Processing on Low-Cost Clusters With GAMMA Active Ports*, Elsevier Science, Parallel Computing 26, 2000, pp. 333-354
[30] M. Lobosco, V. S. Costa, C.L. de Amorim, *Performance Evaluation of Fast Ethernet, Giganet and Myrinet on a Cluster*, International Conference on Computational Science, ICCS 2002, Proceedings, The Netherlands, 2002, pp. 296-305
[31] A. Geist, A. Beguelin, J. Dongarra,W. Jiang, R. Manchek, and V. S. Sunderam, *Parallel Virtual Machine – A User's Guide and Tutorial for Networked Parallel Computing*, MTI Press, London, UK, 1994
[32] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI–The Complete Reference*, Volume 1 - The MPI-1 Core, 2nd edition. The MIT Press, 1998.
[33] J. S. Vetter and F. Mueller, *Communication Characteristics of Large-Scale Scientific Applications for Contemporary Cluster Architectures*, IPDPS, April 2002.
[34] U. Meyer et al. (editors), *Algorithms for Memory Hierarchies*, LNCS 2625, Springer-Verlag, Berlin, 2003, pp. 320-354
[35] I. Foster, *Designing and Building Parallel Programs,* Addison-Wesley Inc., Argonne National Laboratory, USA, 1995. http://www.mcs.anl.gov/dbpp/
[36] D. R. Helman, D. A. Bader, J. Jájá, *Parallel Algorithm for Personalized Communication and Sorting with Experimental Study*, Technical Report CS-TR-3549, Institute for Advanced Computer Studies, University of Maryland, 1995.
[37] J.W. Baugh Jr. and R.K.S. Konduri, *Discrete Element Modeling on a Cluster of Workstations*, Engineering with Computers (2001) 17, Springer-Verlag, London, pp. 1-15.
[38] Li Kuan Ching, Jean-Luc Gaudiot, and Liria Matsumoto Sato, *Performance Prediction Methodology for Parallel Programs with MPI in NOW Environments*, 4th International Workshop on Distributed Computing, Proceedings, Calcutta, India, 2002, pp. 268-279
[39] B, Wilkinson, M. Allen, *Parallel programming,* Prentice Hall, Upper Saddle River, New Jersey, USA, 1999.
[40] Z. Juhász, *An Analytical Method for Predicting the Performance of Parallel Image Processing Operation*, Kluwer Academic Publishers, Boston, 1996. pp. 1-19
[41] David A. Bader, Joseph Jájá, *SIMPLE: A Methodology for Programming High Performance Algorithms on Clusters of Symmetric Multiprocessors*, Institute for Advanced Computer Studies, University of Maryland, 1997.
[42] G.M. Amdahl, *Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities*, AFIPS Conference Proceedings vol. 30 (Atlantic City, N.J., Apr. 18-20). AFIPS Press, Reston, Va., 1967, pp. 483-485.
[43] Z. Xu, K. Whang, *Modeling Communication Overhead: MPI and MPL Performance on the IBM SP2*, IEEE Parallel and Distributed Technology, Vol. 4, No. 1, 1996, pp. 9-24
[44] P. Cremonesi and C. Gennaro, *Integrated Performance Models for SPMD Applications and MIMD Architectures*, IEEE Transactions on Parallel and Distributed Systems, Vol. 13, No. 7., 2002, pp. 745-757
[45] SETI@home, Search for Extraterrestrial Intelligence, Berkeley University of California, USA, http://setiathome.ssl.berkeley.edu/
[46] K. Pearson, *Internet Distributed Computing Projects*, http://www.nyx.net/kpearson/distrib.html
[47] *wxWindows Homepage*, http://www.wxwindows.org
[48] G.F. Pfister, *In Search of Clusters* (Second Edition), Prentice Hall, Upper Saddle River, New Jersey, USA, 1998.