Tallinna Tehnikaülikool
Automaatikainstituut

Taivo Lints
990849LAS

Paralleelarvutused. Arvutiklastrid. Beowulf.

Referaat aines "Arvutivõrgud – projekt"
Õppejõud: Toomas Tommingas

2002. detsember

Sisukord

Piltide indeks

Sissejuhatus

Üha sagedamini võib leida artikleid ja lugusid võrku ühendatud arvutitest, mis koostööd tehes moodustavad ühtse kogumi – klastri. Sellist arvutikobarat iseloomustab ühest küljest kõrge jõudlus, mis teatud juhtumitel on võrreldav isegi superarvutite omaga. Teisalt aga paistab silma rahaline efektiivsus – sama jõudluse korral on klastri koguhind oluliselt madalam kui suurarvutil. Sellised omadused tekitasid käesoleva töö autoris huvi teemaga lähemalt tutvuda ning jõuda selgusele, kas oleks kasulik ka ise (lähemas) tulevikus oma arvutuslike probleemide lahendamiseks klastreid luua ja rakendada.

Üks autorit huvitavaid valdkondi on tehiselu ja -intelligents. Olen seisukohal, et nimetatud valdkonnas on perspektiivikas kasutada üksteisega suhtlevaid alamkomponente, millede koostöö tulemusena võivad avalduda elule ja ka intelligentsile iseloomulikud tunnused. Sellisest lähenemisest tulenevad arvutiprogrammid ja riistvara on realiseeritav tugevalt paralleelsena, st. alamkomponendid tegutsevad samaaegselt, üksteisega paralleelselt. See aga viitab klastrite sobivusele antud programmide jooksutamiseks. Lisaks võib arvutikobarate uurimine anda uusi ideid tehiselusüsteemi alamkomponentide koostööprobleemide lahendamiseks.

Referaadi kirjutamise eesmärk oli eelkõige autori enda harimine paralleelarvutuse alal. Kuna praktilised kogemused arvutikobaratega puuduvad, siis käesolev töö põhineb eranditult Internetist saadaolevatel materjalidel. Tekstis ei ole viiteid allikatele eraldi välja toodud, sest see oleks kohati hakanud lugemist segama, küll aga võib kasutatud materjalide loetelu leida töö lõpust.

Tõlkeküsimuste lahendamisel on prioriteediks valitud arusaadavus, mitte keelearendus. Kui eestikeelne termin on vähelevinud (või puudub), siis on esitatud kas mõlemad keeled või ainult inglise keel.

Paralleelarvutused

Paralleelarvutamise *(parallel processing)* idee on selles, et programm jagatakse mitmeks osaks, mis töötavad samaaegselt – igaüks oma protsessori peal. Tulemuseks on programmi töö kiirenemine. Programm, mida jooksutatakse paralleelselt *n* protsessoril, võiks töötada *n* korda kiiremini võrreldes juhuga, kui kasutatakse ainult ühte protsessorit. Reaalselt muidugi nii head efekti üldjuhul ei saavutata, kuigi teatavad algoritmid võivad tulemuse saavutada isegi rohkem kui *n* korda kiiremini *(super-linear speedup)*.

Konkreetse probleemi lahendamisel on oluline alustada küsimusega: kas meil üleüldse on vaja paralleelarvutust? Näiteks kasutada tekstiredaktori jooksutamiseks 8 protsessorit tundub mõningase liialdusena – ja seda ta ka kindlasti on. Aga kui tegu on renderdamisega (pildi ruumiline viimistlemine), võrguserveriga, andmebaasidega? Siin võib lisaprotsessoritest juba abi olla. Keerukad simulatsioonid, andmekaevandamine – sellistes valdkondades on paralleelarvutus kindlasti abiks. Üldjuhul võib öelda, et paralleeltöötluse kasutamist tasub kaaluda, kui:

- Rakendatav algoritm sisaldab piisavalt paralleelsust ja suudab lisaprotsessorite kasutamisest kasu lõigata. Programmist tuleb leida osad, mis on võimelised iseseisvalt ja samaaegselt eraldi protsessoritel töötama. Selliste osade leidmine ei taga siiski paralleelsuse kasutamisel kõrget efektiivsust. Näiteks programm, millel kulub ühe arvuti peal tulemuste saamiseks neli sekundit, võib küll olla suuteline jooksma neljal protsessoril osadena, millest igaüks vajab ainult sekundi, aga kui nende masinate töö koordineerimiseks on vaja kolm või enam sekundit, siis mingit kasu sellest paralleelsusest ei ole.
- Vajatavast rakendusest on juba saadaval paralleelvariant või te olete valmis selle loomise nimel ise uut või lisakoodi kirjutama.
- Olete huvitatud paralleelarvutamise teaduslikust uurimisest või lihtsalt soovite vastava valdkonnaga tutvuda. Paralleelarvutuste realiseerimine ei ole tingimata alati keeruline, aga enamusele arvutikasutajatele on see siiski tundmatu ala ja raamatut *"Parallel Processing for Dummies"* ei ole vähemalt senimaani veel välja antud.

Arvutusvõimsust vajavad alad, kus on olemas nõudlus paralleelsuse järele, võib jagada kaheks:

- Tootmine (ja ka äritegevus) – andmeid töödeldakse reaalajas, seega on olulised näitajad õigeaegsus, kättesaadavus, töökindlus, usaldatavus, veakindlus.
- Uurimis- ja arendustöö – enamasti vajatakse väga suurt arvutusvõimsust. Käesolevas töös ongi põhirõhk just sellel vajadusel.

Andmaks ettekujutust erinevatest võimalustest kiirete arvutite ja arvutikobarate kasutamiseks on järgnevalt toodud peamised valdkonnad, mis sellist ressurssi vajavad. Suur osa superarvutite TOP500'sse kuuluvaid masinaid jooksutavad justnimelt selliste valdkondade rakendusprogramme.

Ennustav modelleerimine ja simulatsioonid. Atmosfääri, maapealse keskkonna, kosmose ja maailma majanduse multidimensionaalne modelleerimine on valdkonnad, mis pakuvad huvi suurele osale maailma teadlastest. Selline modelleerimine eeldab suuremahulisi arvutusi, et saavutada tulemuste piisavat täpsust. Rakenduste hulka kuuluvad: arvutuslik ilmaennustamine, pooljuhtide simuleerimine, okeanograafia, astrofüüsika (nt. galaktikate kokkupõrke või mustade aukude uurimine), inimgenoomi järjestamine/sekveneerimine, sotsioökonoomilised uuringud.

Tehniline disain ja automatiseerimine. Rakendusi: lõplike elementide analüüs, arvutuslik aerodünaamika, kaugseire, tehisintelligents ja automatiseerimine, sh. pildi- ja kõnetuvastus, robootika, ekspertsüsteemid, CAD/CAM/CAI.

Energiaressursside uurimine. Energia kättesaadavus mõjutab otseselt kogu maailma majanduslikku progressi, mistõttu sellesse valdkonda on suunatud suurel hulgal investeeringuid. Rakendusi: seismiline seire (nafta ja gaasi leidmise eesmärgil), veehoidlate modelleerimine, termotuumareaktsiooni simuleerimine, tuumareaktorite ohutuse uurimine.

Meditsiin, sõjavägi, alusuuringud. Arstiteaduses on kiireid arvuteid vaja näiteks kompuutertomograafias, diagnostikas, tehiselundite simuleerimiseks, ajukahjustuste ennustamiseks ja geneetikas. Sõjavägi kasutab superarvuteid relvade väljatöötamisel, rünnaku mõjude simuleerimisel jms. Kõikvõimalikud alusuuringud nõuavad peaaegu alati suurt arvutusvõimsust, näiteks polümeeride või kvantmehaanika uurimiseks.

Visualiseerimine. Paljud filmid (nt. *The Matrix, Titanic, Toy Story)* sisaldavad tohutul hulgal detailirikast arvutianimatsiooni ning nende filmide mõistliku ajaga valmimist võimaldas eelkõige arvutiklastrite kasutamine. Reaalsete dekoratsioonide asendamine arvuti poolt genereerituga võimaldab hoida kokku raha, rääkimata võimalusest rikkuda füüsikaseadusi. Arvutigraafika on vajalik ka kõikvõimalike andmete visualiseerimiseks. Hiiglaslike andmemahtude korral on paralleeltöötlus paratamatus.

Järgmine küsimus, mida tihti esitatakse, on: "Milleks mulle kaks või neli protsessorit, parem ootan veidi, kuni müügile tuleb 986 turbohüperprotsessor." Vastus:

- Protsessorite kiirused küll kahekordistuvad keskmiselt iga 18 kuuga, aga mälu ja kõvaketaste kiirused mitte. Enamus programme suhtlevad töö käigus erinevate mäluseadmetega, eriti RAM'i ja kõvakettaga, mistõttu nende aeglus võib saada piiravaks teguriks. Paralleeltöötlus on üks võimalusi selle probleemi lahendamiseks.
- Ennustatakse, et protsessorite kiiruse kasvutempo aeglustub mõne aasta pärast tehnoloogiliste ja füüsikaliste piirangute tõttu, kui just ei võeta kasutusele põhimõtteliselt teistsuguseid tehnoloogiaid.
- Sõltuvalt rakendusest võib paralleelarvutus töökiirust suurendada isegi kuni sadu kordi. Üksik protsessor sellist jõudlust ei võimalda. Isegi superarvutid, mis varem kasutasid väga kiireid spetsiaalprotsessoreid, ehitatakse nüüdisajal enamasti suurest hulgast masstootmises olevatest protsessoritest.

- Mitmetes rakendustes ei olegi peamine kiirus, vaid näiteks hoopis töökindlus, mille tagamise üks võimalusi on kasutada mitmest arvutist koosnevat süsteemi, kus ühe osa riknemine ei häiri oluliselt kogu süsteemi tööd – ülejäänud osad võtavad riknenu funktsioonide täitmise enda peale.

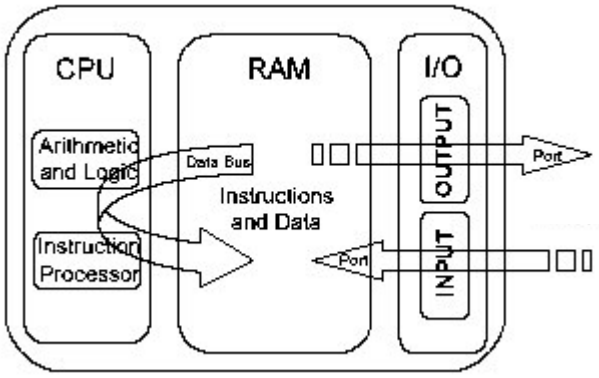
Arvutite klassifikatsioon

Vastavalt Michael J. Flynn'i 1966 aastal väljapakutud arvutiarhitektuuride liigitusele jagunevad arvutid nelja klassi:

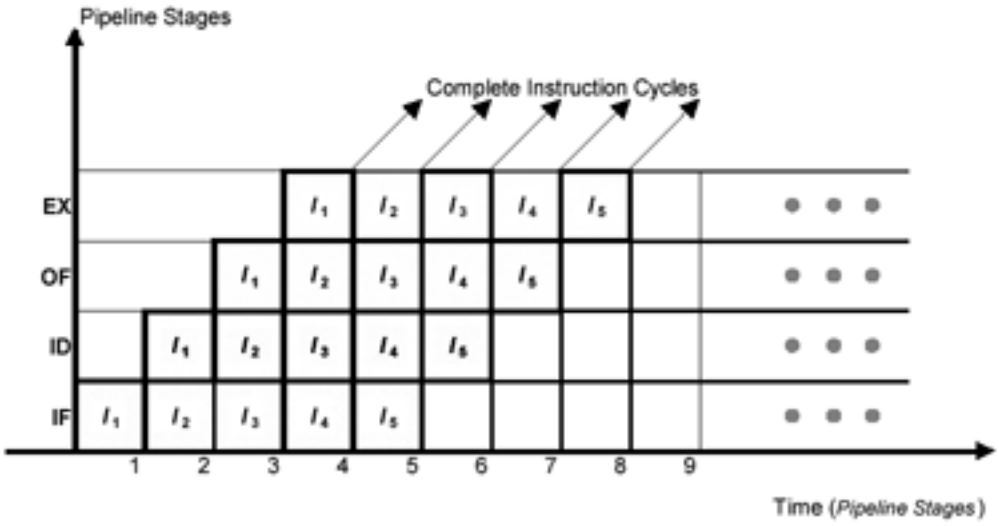
- **SISD** – *Single Instruction stream / Single Data stream*
- **SIMD** – *Single Instruction stream / Multiple Data stream*
- **MISD** – *Multiple Instruction stream / Single Data stream*
- **MIMD** – *Multiple Instruction stream / Multiple Data stream*

Instruction stream = käskude voog; *Data stream* = andmete voog.

SISD – käske täidetakse järjestikuliselt ja rakendatakse ühele andmevoole (kuigi nende täitmisel võib esineda mõnigast paralleelsust protsessori sees – konveier-töötlust *(pipelining)*). Kui SISD arvuti sisaldab mitut funktsionaalset alamüksust, siis kõik need on ühe juhtseadme kontrolli all. Sellesse klassi kuulub enamik tavakasutuses olevaid arvuteid.



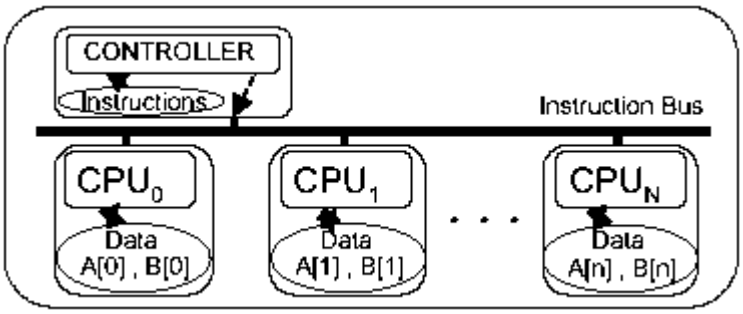
J1. Ühe protsessoriga arvuti, SISD.



J2. Konveiertöötlus protsessori sees.

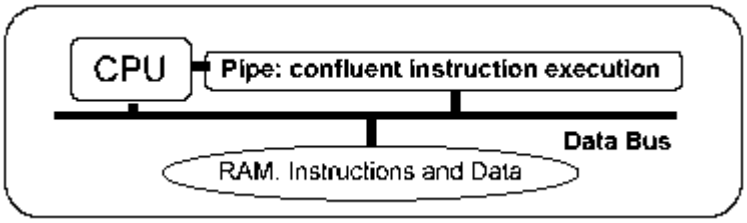
(IF – Instruction Fetch; ID – Instruction Decoding; OF – Operand Fetch; EX – Execution)

SIMD – sama käsku rakendatakse korraga mitmele andmevoole. Selliseid arvuteid nimetatakse maatriks- ja vektorarvutiteks.



J3. SIMD arhitektuur.

MISD – ühele andmevoole rakendatakse korraga erinevaid käske. Mõnedes käsitlustes loetakse sellist klassi ainult teoreetiliseks variandiks, teistes asetatakse siia eelpoolmainitud konveiertöötlus.



J4. Konveiertöötlus kui MISD.

MIMD – nii käsu- kui andmevooge on rohkem kui üks. Sellesse klassi kuulub suurem osa multiprotsessor- ja multiarvutisüsteeme. MIMD peamised omadused on:

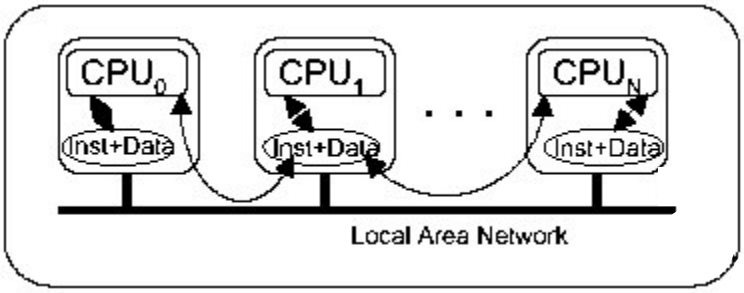
- Iga protsessor käitub vastavalt isiklikule käsuvoole.
- Iga protsessor lahendab erinevat osa tervikprobleemist.
- Iga protsessor edastab andmeid ka teistele.
- Protsessorid peavad aeg-ajalt teisi järele ootama või seisma järjekorras, et pääseda ligi andmetele.

Kui andmevood omavahel seotud ei ole, siis on tegu MSISD süsteemiga *(Multiple SISD)*, mis ei ole midagi muud kui hulk iseseisvaid SISD arvuteid. Üldjuhul aga moodustab MIMD süsteem ühtse terviku ja andmevahetus protsessorite vahel on tihe. Siit järeldub, et sellise süsteemi väga oluline komponent on sidesüsteem protsessorite vahel. Olenevalt arhitektuurist võib suhtluskanaliks olla näiteks (super)arvutisisene siin või hoopis arvutitevaheline võrk.

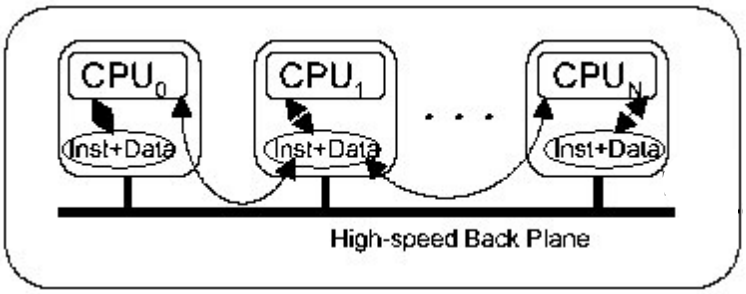
Protsessorite omavahelise suhtluse intensiivsuse järgi saab MIMD süsteemid jagada kaheks:

Nõrgalt seotud süsteemid. Igal protsessoril on teatav hulk isiklikke sisend- / väljundseadmeid ja suur kohalik mälu, kus asub enamus temale vajalikke käske ning andmeid. Erinevate arvutimoodulite (protsessorid koos isikliku mälu ja s/v-seadmetega) peal jooksvad protsessid suhtlevad omavahel sõnumite abil. Sellised süsteemid on efektiivsed eelkõige juhul, kui protsessidevaheline suhtlus ei ole

väga tihe. Kui aga rakendus vajab väga väikseid kosteaegu *(response time)*, võib nõrgalt seotud süsteem osutuda ebasobivaks.

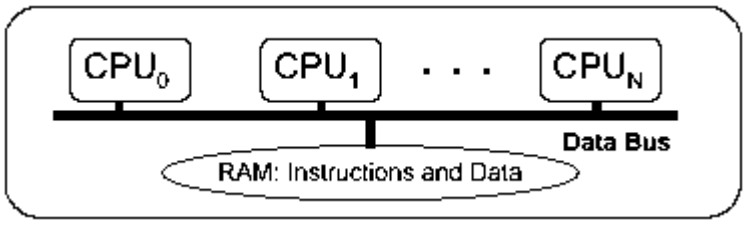


J5. Nõrgalt seotud MIMD.
Eraldiseisvad arvutid suhtlevad üle LAN'i. Nt. Beowulf.



J6. Nõrgalt seotud MIMD.
Näiteks protsessori ja mäluga arvutikaardid,
mis suhtlevad üle kohaliku siini.

Tihedalt seotud süsteemid. Protsessorid suhtlevad läbi ühise mälu. Seega protsessoritevahelise suhtluse kiirus ulatub mälu kasutamise kiiruseni ja saavutatav jõudlus on suurem kui nõrgalt seotud süsteemidel. Suurem on ka riistvara maksumus. Sellesse klassi kuuluvad näiteks SMP masinad *(Symmetric Multiprocessing)*.



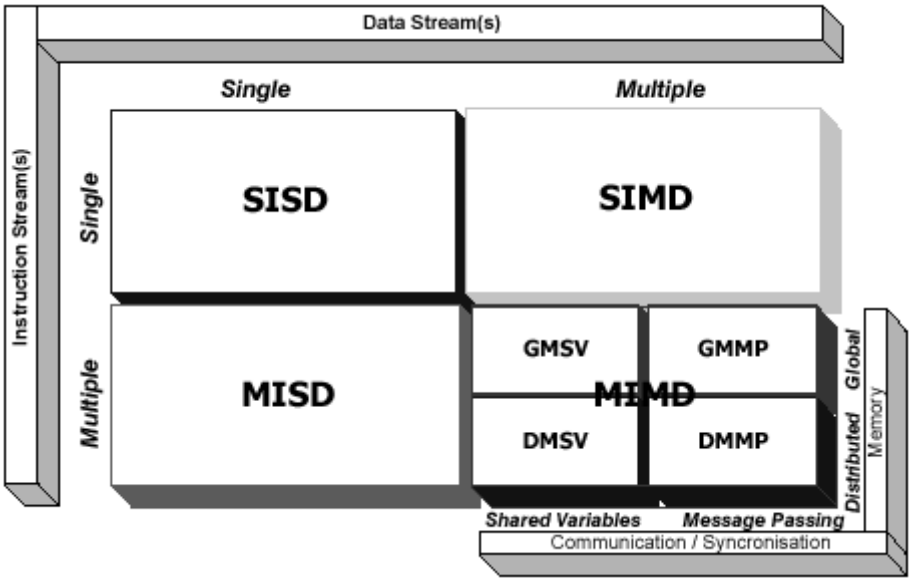
J7. Tihedalt seotud MIMD.

Ilmneb, et MIMD kategooria sisaldab suurel hulgal erinevaid arvutiarhitektuure. Seetõttu, vastavalt E. E. Johnson'i ettepanekule (1988) liigitatakse see kategooria omakorda neljaks. Liigituse aluseks on mälustruktuur (globaalne või hajus) ja suhtlusmehhanism (ühised muutujad või sõnumivahetus).

- **GMSV** – *Global Memory / Shared Variables.*
- **GMMP** – *Global Memory / Message Passing* (tavaliselt ei kasutata).
- **DMSV** – *Distributed Memory / Shared Variables.*
- **DMMP** – *Distributed Memory / Message Passing.*

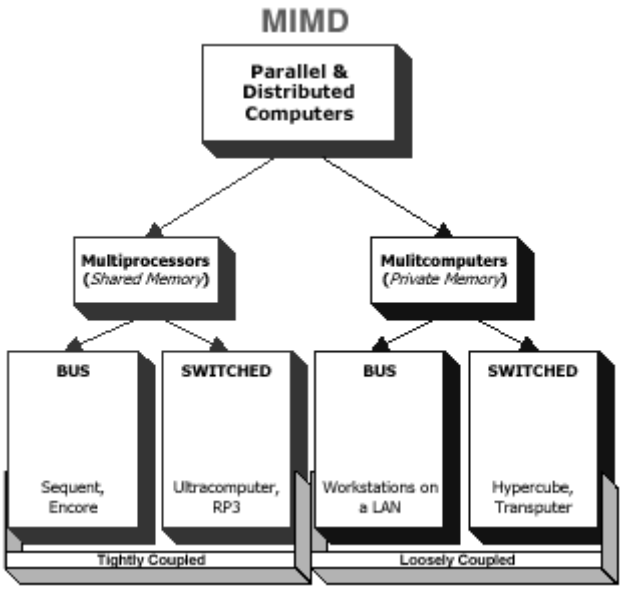
Enamus klastreid liigitub DMMP alla.

Viies kogu eelneva liigituse ühele pildile kokku, saame Flynn-Johnson'i klassifikatsiooniskeemi arvutisüsteemide kohta:



J8. Flynn-Johnson'i klassifikatsioon.

Alternatiivne võimalus MIMD kategooriat osadeks jagada on suhtlusmeetodi asemel arvestada võrguarhitektuuri:

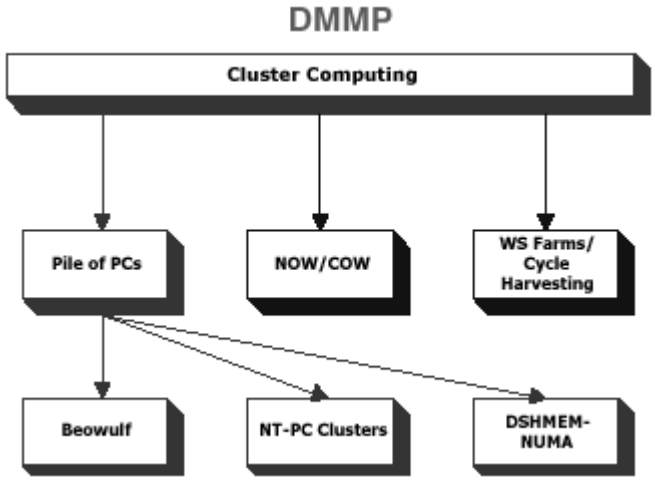


J9. Tannenbaum'i klassifikatsioon.

Märkus: joonisel on LAN' iga ühendatud tööjaamad paigutatud ainult *bus*-tüüpi võrkude alla, kuid kuna *switch*'ide hinnad on järjest soodsamad, leiavad need enam kasutust ka tööjaamade / PC' de ühendamisel kobarateks.

Klastrite klassifikatsioon

Nagu eelnevalt mainitud, klassifitseeruvad klastrid üldjuhul DMMP *(Distributed Memory / Message Passing)* alla. Aga klastreid saab ka omakorda klassideks jagada. Üks võimalikest variantidest on järgnev:



J10. Klastrite klassifikatsioon.

NOW / COW = Network / Cluster of Workstations; WS = Workstation
NUMA = Non-Uniform Memory Access, st. arvutid jagavad üksteisega mälu, mille tulemusena mälupöördusaeg on muutuv suurus ja sõltub vajatava mälu füüsilisest asukohast.

WS Farms / Cycle Harvesting kujutab endast katset kasutada võrku ühendatud tavaliste tööjaamade vabu ressursse, sest töötajad ise oma arvuteid üldjuhul maksimaalselt ei koorma. Selline keskkond nõuab äärmiselt tolerantseid algoritme, mida ei häiri suured viited andmevahetusel ja ebaühtlane ning pidevalt muutuv koormusjaotus arvutite vahel.

NOW / COW kasutab samuti võrku ühendatud tööjaamu, kuid tavaliselt sellistel aegadel, mil töötajad ise oma arvuteid ei kasuta – öösel ja nädalavahetustel.

Beowulf tüüpi klaster erineb eelnevatest eelkõige selle poolest, et tavaliselt puuduvad arvutussõlmedel (PC' d) eraldi klaviatuurid, hiired, videokaardid ja monitorid – Beowulf' i masin on spetsiaalselt ehitatud paralleelarvutuste jaoks ja vastavalt ka optimeeritud. Kuid see ei ole siiski range reegel. Erinevus *NUMA*' dest seisneb suhtlusviisis – puudub ühine suur mäluruum, mille moodustavad kõigi osalejate mälud. Erinevus NT-klastritest seisneb operatsioonisüsteemis: Windows' i asemel kasutatakse Linux' it.

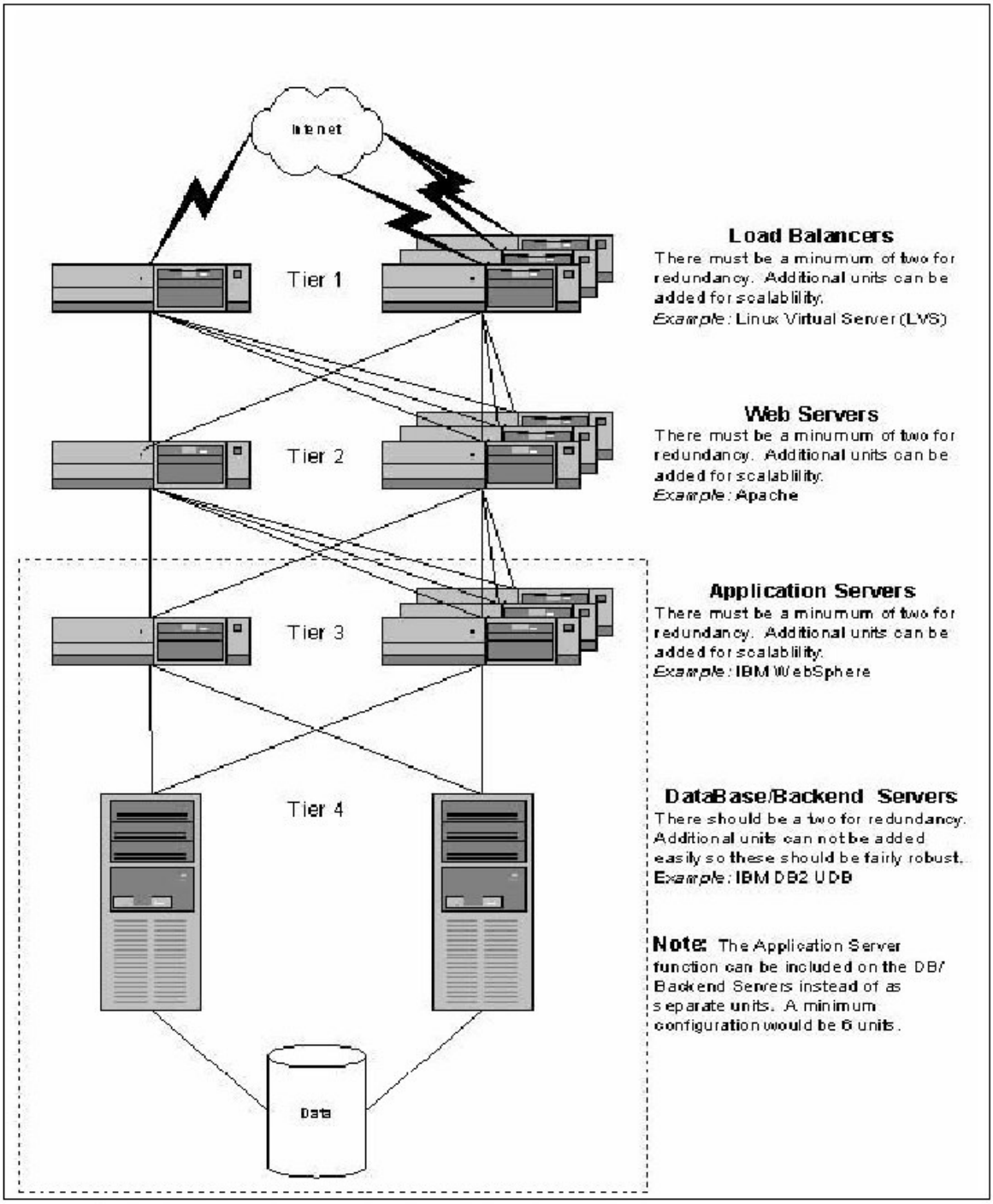
Beowulf klastreid jagatatakse kaheks. *Class I Beowulf* koosneb eranditult masstootmises olevatest laialt levinud komponentidest *(COTS – Commodity Off The Shelf)*. Tüüptestiks on tingimus, et *Class I* masinat peab olema võimalik kokku panna osadest, mida leidub vähemalt kolmes suuremas arvutikaupade reklaamkataloogis. *Class II Beowulf* on masin, mis eelnevat tingimust ei rahulda. *Class II* eeliseks on üldjuhul parem jõudlus, sest kasutatakse paremaid komponente, kuid sellest tulenevalt on kõrgem ka hind ning võib tekkida sõltuvus (hinna, riistvara ja draiverite kättesaadavuse osas) ühest konkreetsest riistvaratootjast, kes neid

spetsiaalkomponente toodab. Kumbki klass ei ole teisest tingimata parem. See sõltub konkreetsest rakendusest ning nõutavast hinna / jõudluse suhtest.

Arvutikobaraid võib liigitada ka lähtuvalt funktsionaalsusest: kõrge töövalmidusega *(HA – High Availability)* klastrid ja suure jõudlusega klastrid *(HPC – High Performance Computing)*. Alati ei ole võimalik nende kahe vahel selget piiri tõmmata, sest vaja võib minna nii suurt jõudlust kui ka veakindlust ja kättesaadavust, kuid oluline ei olegi mitte range klassifitseerimine, vaid erinevate jaotusvõimaluste esiletoomise kaudu oluliste aspektide valgustamine.

Suure jõudlusega klastrite eesmärke oli eespool juba kirjeldatud (vt. ptk. "Paralleelarvutused"). Seetõttu on järgnevas lühidalt iseloomustatud kõrge töövalmidusega klastreid.

Tüüpilises HA kobaras on kaks või enam masinat, mis peegeldavad üksteise funktsioone. Selle saavutamiseks on kaks levinud meetodit. Esimene: üks masin vaikselt jälgib teist ja on valmis teise kokkujooksmise korral tööd üle võtma. Teine: mõlemad masinad on aktiivsed, kusjuures mõlema koormus hoitakse alla 50%, et ühe kokkujooksmisel teine suudaks töö täielikult enda peale võtta. Enamasti need masinad jagavad omavahel ühte kettamassiivi. Veakindluse tõstmiseks võib ka kettamassiive olla kaks, kuid sel juhul tuleb garanteerida andmete ühesus. Eriti ühe kettaseadme korral on tarvis ka mehhanismi, millega korras arvuti saab riknenud masina "ohutuks muuta", et see andmeid ohustama ei hakkaks.



J11. Võrguklastri põhimõte.

Üks levinud HA rakendusi on võrguklastrid ja võrgufarmid. Sellised kobarad on tihti realiseeritud paljusid erinevaid tehnoloogiaid ja arvutiplatvorme kasutades. Tüüpiline võrguklaster on pigem kogum masinaid, mis loob infrastruktuuri, kui lihtsalt klaster. Sellise struktuuri üks ots (vt. J11) on seotud Internetiga ja teises otsas asuvad andmed. Komponentide lühikirjeldused on antud joonisel. Nagu näha, on komponentide vahel suurel hulgal ühendusi. Need suurendavad veakindlust ning samuti leiavad kasutust koormuse jagamisel. Reaalsuses võib muidugi joonisel toodud erinevaid kihte kombineerida ühte masinasse või masinapaari, sõltuvalt nõutavast töökindlusest ja olemasolevatest finantsidest nende nõuete rahuldamiseks.

Klastri komponendid

Edasises käsitluses on klastrite all mõeldud eelkõige suure jõudlusega arvuti-kobaraid. Selliste klastrite peamised komponendid on:

- Arvutid (PC' d, tööjaamad, SMP masinad).
- Võrgud, mis arvuteid ühendavad.
- Operatsioonisüsteem.
- Vahetarkvara *(middleware)* – integreerib eraldi arvutid omavahel nii, et kasutajale jääks mulje terviklikust süsteemist *(Single System Image)*.
- Paralleelprogrammeerimise keskkonnad.
- Rakendused.

Arvutid

Klastri moodustavaid arvuteid ehk "sõlmi" *(nodes)* võib üksteisest eristada nende poolt täidetavate funktsioonide järgi:

- Kasutajaga suhtlemine
- Klastri juhtimine
- Klastri haldamine
- Andmete salvestamine
- Arvutussõlmede installeerimine
- Arvutuste sooritamine.

Funktsioonide jagamine reaalsetele arvutitele sõltub eelkõige projekti eelarvest ja eesmärkidest. Lihtsamate kobarate korral võivad kõik eelnimetatud funktsioonid olla esindatud ühes arvutis, suurte ja kallite klastrite korral aga täidavad ühte konkreetset funktsiooni mitu masinat korraga. Viimane funktsioon – arvutuste sooritamine – on siiski ka lihtsate klastrite korral tavaliselt mitme arvuti vahel jagatud, sest kõrge jõudluse saavutamise idee ongi ju töö osadeks jagamine.

Kasutajaga suhtlemine. See arvuti on klastri "aken maailma", mille kaudu kasutaja oma probleeme arvutikobarale lahendada annab. Olenevalt konfiguratsioonist kasutaja kas peab ise füüsiliselt kohal olema või kui see arvuti on ühendatud ette-võtte kohalikku võrku või Internetti ja varustatud vastava tarkvaraga, siis saab klastriga suhelda ka eemalt.

Klastri juhtimine. Juhtsõlme ülesandeks on tööde ja nende osade jagamine arvutite vahel. Samuti tagab ta mitmed klastri tööks vajalikud teenused nagu *DHCP (Dynamic Host Configuration Protocol), DNS (Domain Name System), NFS (Network File System)*.

Klastri haldamine. Haldussõlm haldab klastrit ja tema komponente (*switch*' e jms.), näiteks korjab *SNMP (Simple Network Management Protocol)* alarme.

Andmete salvestamine. Võimsad kobarad kasutavad ja tekitavad töö käigus tohutul hulgal andmeid, mille salvestamiseks kasutatakse spetsiaalseid masinaid.

Arvutussõlmede installeerimine. Installatsioonisõlm on tavaliselt server, mis paigutab arvutussõlmedesse konkreetse töö jaoks vajalikud failid, sh. operatsioonisüsteemi, teegid *(libraries)* ja kogu arvutamiseks vajaliku tarkvara.

Arvutuste sooritamine. Nende sõlmede ülesanne on arvutusi läbi viia, st. teha seda, mida kasutaja tegelikult vajab.

Hea arhitektuuriga klastri korral ei ole ühe arvutussõlme riknemine katastroofiline – tema töö jagatakse teistele. Küll aga oleks soovitav ülejäänud funktsioone täitvad arvutid teha võimalikult töökindlaks, sest neid suurel hulgal dubleerida ei ole mõttekas. Töökindluse tõstmiseks saab kasutada näiteks *RAID*'e *(Redundant Array of Independent Disks)*, lisaventilaatoreid ja lisatoiteallikaid.

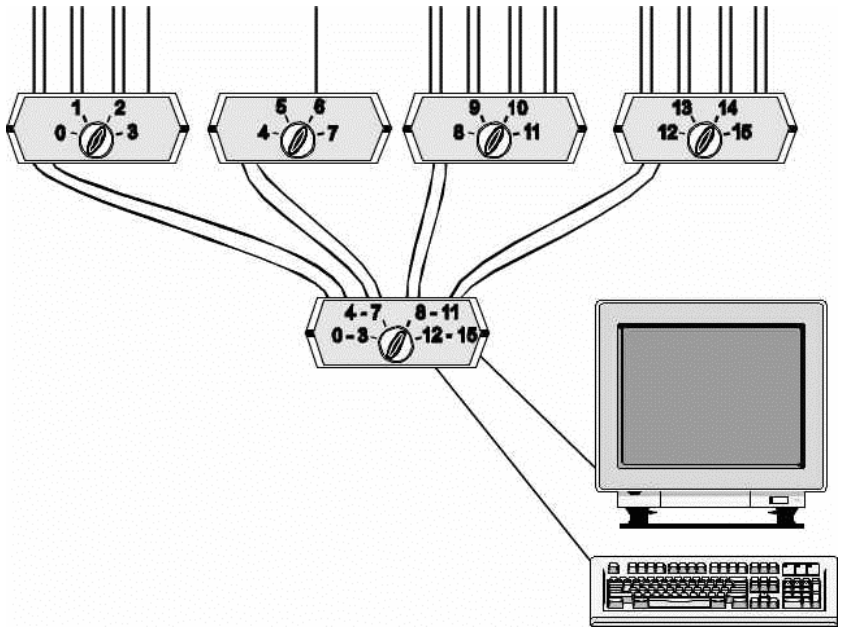
Kui tegu ei ole väga keerukate klastritega, siis arvutid liigitatakse *Master Node*'iks, mis täidab kõiki juhtimise jms. seotud funktsioone, ja S*lave Node*'ideks, mis tegelevad arvutamisega.

Master Node'i valikul on olulisemad kriteeriumid protsessori kiirus, kettaseadme kiirus ja mälu. Kuna *Master* juhib klastri tööd, siis aeglase masina korral kannatab tõsiselt terve kobara töövõime. Eriti oluline on sellega arvestada juhul, kui *Master* täidab lisaks ka veel arvutussõlme funktsiooni. Samuti kasutab terve kobar *Master*'i kõvaketast lähteandmete võtmiseks ja tulemuste salvestamiseks (suurel klastril on selle jaoks spetsiaalseadmed, nagu eespool mainitud), mistõttu peaks kõvaketas olema kiire ja laia andmevahetuskanaliga, näiteks *SCSI (Small Computer System Interface)*. Suur mälu vähendab vajadust mällu mittemahtuvat infot ajutiselt kettale kirjutada; samuti võimaldab kettale kirjutamiste arvu vähendada, hoides info nende kirjutamiste kohta senikaua mälus, kuni need operatsioonid tõepoolest reaalselt on vaja läbi viia.

Slave Node'ide korral on olulised protsessori parameetrid ja mälu. Lisaks protsessori kiirusele on olulised ka tema vahemälude omadused. Hinna ja jõudluse suhte järgi on eelistatumad AMD protsessorid, kuid seejuures tuleb arvestada, et need tarbivad rohkem võimsust ja eraldavad rohkem kuumust, mis suure hulga protsessorite korral võib olla tõsiseltvõetav probleem (nii klastri stabiilsusele kui operaatori enesetundele), nõudes tugevamat ventilatsiooni. Hinna ja jõudluse suhte poolest on soodsam kasutada mitme protsessoriga arvuteid (*SMP* masinaid), kuigi puhtalt jõudluse seisukohast võib üksikprotsessoritest koosnev klaster efektiivsem olla. *SMP* masinad vajavad ühe protsessori kohta vähem võrguühendusi (ja seega ka vähem *switch*'e), arvutikorpuseid, ruumi, voolu. Mis puutub kettaseadmetesse, siis mõne lahenduse korral neid *Slave Node*'idel üldse ei olegi.

Loomulikult ei ole eelpool toodud näitajad ainukesed olulised. Vaadelda tuleb siiski arvutite kui terviksüsteemide jõudlust, mida mõjutavad muuhulgas ka siinide kiirused jms.

Kui arvutikobar on kokku pandud tavalistest PC' dest või tööjaamadest, milledel ei ole monitore, klaviatuure ega hiiri (mis on tüüpiline Beowulf klastrites), siis on kasulik osta *KVM (Keyboard Video Mouse Switch)*, mis võimaldab lihtsat ligipääsu igasse üksikusse arvutisse. Alternatiivne variant on kasutada käru koos monitori, klaviatuuri ja hiirega *(Crash Cart)* ning konkreetse sõlme juurde ise kohale sõita.



J12. Kuidas ühendada KVM' e.

Võrgud

Arhitektuur

Justnimelt arvutivõrk on see, mis võimaldab iseseisvad arvutid muuta ühtseks süsteemiks. Kõrge jõudlusega klastris on protsessorid tavaliselt üksteisega tihedas infovahetuses ning seetõttu on klastri loomisel äärmiselt oluline valida sobivad meetodid ja vahendid arvutite omavaheliseks ühendamiseks.

Võrgu disainimisel tuleb kõigepealt arvesse võtta, millised rakendused sellel klastril jooksma hakkavad:

- Kui tihe on rakenduse töötamise ajal suhtlemine võrgusõlmede vahel.
- Millised tagajärjed on suhtlusele mõjuvatel piirangutel klastri töövõimele.

Kui need omadused on kindlaks tehtud, siis on võimalik hinnata jõudlusnõudeid võrgule, valida võrgu topoloogia ja seejärel tehnoloogia.

Võrgu topoloogiat iseloomustavad parameetrid on näiteks:

- Võrgu diameeter – kõige pikem lühimatest teedest kahe (suvalise) võrgusõlme vahel. Peaks olema suhteliselt väike, kui eesmärgiks on minimiseerida viiteid / latentsust *(latency)*. Pakettide edastamisel *store-and-forward* meetodiga (vahepealsed sõlmed salvestavad sõnumi tervikuna ja alles siis saadavad edasi) on võrgu diameetri mõju märksa suurem võrreldes *wormhole* marsruutimisega (kus sõnumi sisenevad osad suunatakse koheselt edasi ilma terviku kohalejõudmist ootamata).
- *Bisection (band)width* ("poolituskoha" (riba)laius) – vähim number ühendusi (või nende summaarne ribalaius), mis tuleb läbi lõigata, et jagada võrk kaheks võrdseks osaks. See on oluline, kui sõlmed suhtlevad üksteisega suvaliselt, mitte ainult mõne konkreetse (naaber)sõlmega. Kitsas poolituskoht piirab kahe poole omavahelist suhtlemist ja seetõttu mõjub halvasti tihedat suhtlust vajavate algoritmide jõudlusele.
- Sõlmede järk *(vertex / node degree)* – igas sõlmes vajatavate kommunikatsiooniportide arv. Peaks olema konstante, mitte sõltuma võrgu suurusest (kui me soovime saada kergesti laiendatavat võrku). Sõlmede järk mõjutab otseselt iga sõlme maksumust, kusjuures mõju on suurem mitut juhet sisaldavate paralleelportide korral või kui sõlm peab suutma suhelda läbi kõigi portide korraga.

Järgnevas on illustratsiooni mõttes toodud nende kolme parameetri väärtused mõnedel levinumatel võrgutopoloogiatel. See nimekiri ei ole mingil juhul täielik, kuid annab ettekujutuse nende parameetrite suurest varieeruvusest.

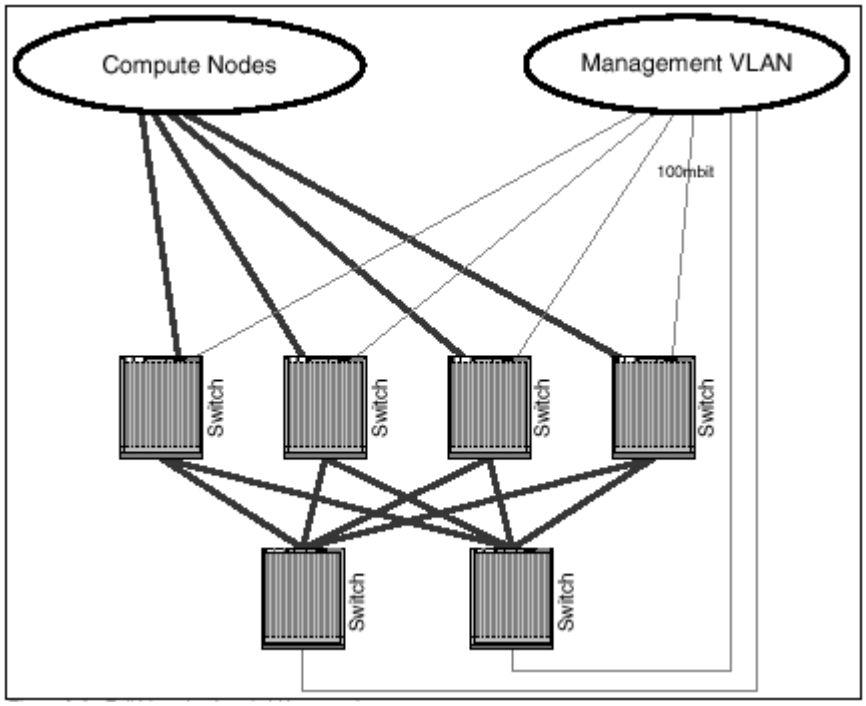
Network Topology	No. of Nodes	Network Diameter	Bisection Width	Node Degree	Local Links
1D Mesh (linear array)	k	k–1	1	2	Yes
1D Torus (ring, loop)	k	k/2	2	2	Yes
2D Mesh	k^2	2k–2	k	4	Yes
2D Torus (k-ary 2 cube)	k^2	k	2k	4	Yes[1]
3D Mesh	k^3	3k-3	k^2	6	Yes
3D Torus (k-ary 3 cube)	k^3	3k/2	$2k^2$	6	Yes[1]
Pyramid	$(4k^2-1)/3$	$2\log_2 k$	2k	9	No
Binary Tree	$2l-1$	$2l-2$	1	3	No
4-ary hypertree	$2^l(2^{l+1}-1)$	$2l$	2^{l+1}	6	No
Butterfly	$2^l(l+1)$	$2l$	2^l	4	No
Hypercube	2^l	l	2^{l-1}	l	No
Cube-connected cycles	$2^l l$	$2l$	2^{l-1}	3	No
Shuffle-exchange	2^l	$2l-1$	$\geq \frac{2^{l-1}}{l}$	4 unidir	No
De Bruijn	2^l	l	$\frac{2^l}{l}$	4 unidir	No

[1] With Folded Layout

T1. Mõningate võrkude topoloogilisi parameetreid.

Tabelis toodud ühendusvariante on kasutatud paljudes paralleelsüsteemides, aga väiksemad süsteemid kasutavad enamasti siinil *(bus)* põhinevat arhitektuuri. Kuna üheainsa siini kasutamine muutub protsessorite arvu kasvades kiiresti peamiseks piiravaks teguriks, kasutatakse erinevaid mitme siiniga arhitektuure ja hierarhilisi skeeme. Seoses hinna langemisega leiavad järjest enam kasutust *switch*' id, mis erinevalt *hub*' ist ei saada sissetulnud paketti kõigile sõlmedele laiali, vaid edastavad paketi sõlmele, kellele see määratud on (paketis leiduva aadressivälja sisu järgi).

Klastri sõlmede arvu kasvades tekib probleem – ühest *switch*' ist jääb väheks. *Switch*' ide lisamisel tuleb aga arvestada, et on vaja garanteerida maksimaalne poolituskoha ribalaius *(full bisection bandwidth)*, kuna iga sõlm peab saama suhelda suvalise teise sõlmega, ilma et ükski marsruuter tee peale ette jääks (sest marsruuterid vähendaks klastri jõudlust). *Switch*' ide ühendamise näide on toodud joonisel J13.

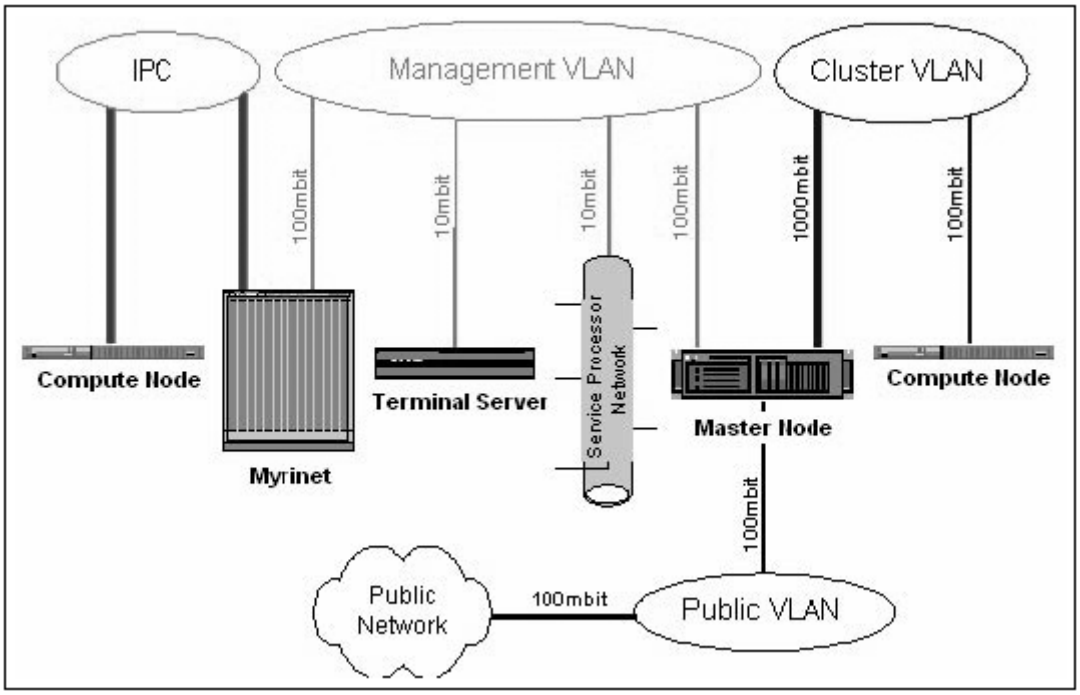


J13. Full bisection bandwidth.

Beowulf tüüpi arvutikobara korral, nagu eelpool mainitud, kasutatakse eelkõige laialt levinud ja odavamaid vahendeid. See aga tähendab suhteliselt aeglaste võrkude kasutamist, mis mõjub klastri jõudlusele halvasti. Selle probleemi lahendamiseks on välja pakutud võrgu kanalite sidumise idee *(Network Channel Bonding)* – mitu väikese maksumusega võrku ühendatakse üheks suurema läbilaskvusega loogiliseks võrguks. Ainus lisanduv töö on arvutuslikult vähenõudlik ülesanne jaotada paketid kasutadaolevate seadmete saatejärjekordadesse laiali. Meetod töötati välja NASA' s Beowulf' i jaoks ja kasutas algul kahte 10Mbps Ethernet' i. Nüüdseks on suurenenud võimalike kanalite arv ja kasutusele on võetud kiiremad võrgud. Rakendus tohib kanalite sidumise korral märgata ainult võrgu jõudluse kasvu, selle saavutamine peab jääma madalamatele tasemetele.

Kanalite sidumise kõrvalproduktiks on teatav võrkude liiasus, kus sõlmi ühendab kaks või rohkem võrku. Seda on võimalik ära kasutada klastri veakindluse suurendamiseks.

Nagu selgus, on võimalik mitut reaalset võrku kokku siduda üheks loogiliseks. Aga kasutatakse ka teistsugust lähenemist – ühest (reaalsest) võrgust tehakse mitu loogilist. See on mõistlik suuremate klastrite puhul, kus kasutatakse väga kiiret võrku ning intelligentseid *switch*' e, mis oskavad virtuaalseid kohtvõrke *(VLAN)* moodustada. Virtuaalvõrkude kasutamine võimaldab saavutada suuremat paindlikkust ja turvalisust ning ka efektiivsemat võrguressursi kasutamist. *VLAN* tehnoloogia tähendab põhimõtteliselt võrgu segmenteerimist erinevateks levialadeks *(broadcast domain)*, nii et pakette kommuteeritakse ainult konkreetsesse *VLAN*' i kuuluvate portite vahel. Üks võimalikest jagamisvariantidest on toodud joonisel J14.



J14. Virtuaalvõrgud klastris.

IPC (InterProcess Communication). Läbi selle virtuaalvõrgu suhtlevad omavahel arvutussõlmed töö käigus.

Management. Seda *VLAN*' i kasutatakse ainult komponentide haldamiseks. Turvalisuse huvideks peaks olema isoleeritud.

Cluster. Arvutus- ja salvestussõlmed kasutavad seda ainult info sisend- / väljund-voogude, *Master* ainult installeerimise ja teatavate haldusprotseduuride jaoks.

Public. Selle virtuaalvõrgu kaudu toimub kasutajate ligipääs klastrile.

Võrgutehnoloogiad

Arvutivõrkude tehnoloogiad arenevad äärmiselt kiiresti. Pidevalt laieneb valik, kasvavad kiirused, langevad hinnad ja muutuvad muud parameetrid. Seetõttu on väga raske anda hetkeseisu iseloomustavat infot. Järgnevas on toodud paljude võrkude tutvustused klastritele sobivuse seisukohast. Kuna refereeritav allikas on ilmselt vähemalt ühe-kahe aasta vanune, siis on välja jäetud seal toodud hinnad, milledel on kalduvus väga palju muutuda (odavamaks). Ülejäänud andmetesse tuleks samuti suhtuda teatava ettevaatusega, aga kui need on ka vahepeal muutunud, siis ikka paremuse poole. Küsimärgiga "?" tähistatud andmetes ei ole ka allika autor kindel. Info vastavusse viimine käesoleva hetkega oleks nõudnud põhjalikumat uurimistööd, mida ajapuuduse ja samuti käesoleva referaadi iseloomu tõttu läbi viima ei hakanud. Nimekirjas kasutatud karakteristikute selgitused:

*Linux' i toetus*Kuna Linux on peamine väikese eelarvega klastrites kasutatav operatsioonisüsteem, siis on väga oluline, kas antud võrgutehnoloogia on Linux' i all kasutatav.

Maksimaalne ribalaius / läbilaskevõime. Number, millele enamasti kõige rohkem tähelepanu pööratakse. Üldjuhul antud teoreetiline parim väärtus.

Minimaalne viide / latentsus. Number, millele peaks klastrite korral isegi rohkem tähelepanu pöörama kui ribalaiusele. Jällegi antud ebarealistlikud parimad väärtused. Üldjuhul võrgu enda latentsus on kõigest mõni mikrosekund. Sellest palju suuremad numbrid näitavad arvukate ebaefektiivsete riist- ja tarkvarakihtide olemasolu võrguliideses.

Kättesaadavus. Üldkasutatavad tehnoloogiad on vabalt kättesaadavad paljudelt tootjatelt. Mitme-tootja-tehnoloogiaid pakub rohkem kui üks konkureeriv valmistaja, kuid pakutav ei ole alati ühtne – võib esineda kokkusobivusprobleeme. Ühe-tootja-tehnoloogia jätab selle kasutaja üheainsa tootja meelevalda (näiteks hinna ning tarkvaralise toe poolest). Ning viimaks vabakasutuses tehnoloogia – isegi kui seda keegi ei müü, võib igaüks osta vajalikud osad ning selle ise valmis teha. Siia kuuluvad eelkõige uurimis- ja arendustöös välja pakutud prototüübid.

Kasutatav port või siin. Parima jõudluse annab tavaliselt PCI siinile ühendatav võrgukaart, aga kohati kasutatakse ka vanemaid siinitüüpe. Liidesed, mis ei nõua arvuti korpuse avamist, on kindlasti meeldivamad ning õnneks ka arenevad kiiresti (SPP *(Standard Parallel Port)*, IrDa, USB, IEE1284, FireWire jms.)

Võrgustruktuur. Kas kasutatakse siiniarhitektuuri vms.

ArcNet

Linux' i toetus*kernel drivers*
Maksimaalne läbilaskevõime: *2,5 Mb/s*
Minimaalne latentsus: *1000 ìs?*
Kättesaadavus: *mitme-tootja-tehnoloogia*
Kasutatav port või siin: *ISA*
Võrgustruktuur: *unswitched hub or bus (logical ring)*

ArcNet on lokaalvõrk, mis on mõeldud kasutamiseks peamiselt reaalaja sardsüsteemides. Füüsiline struktuur sarnane Ethernet'ile, kuid kasutab *token*' il baseeruvat protokolli loogilise ringi moodustamiseks. Paketipäised on väikesed (3 või 4 baiti) ja sõnumid võivad kanda isegi ühtainsat andmebaiti. Tänu sellele on ArcNet' i jõudlusnäitajad stabiilsemad kui Ethernet' il (piiratud viited jms.). Kahjuks on ta Ethernet' ist aeglasem ja vähem populaarne, seetõttu ka kallim.

ATM

Linux' i toetus*kernel driver, AAL library*
Maksimaalne läbilaskevõime: *155 Mb/s, 1200 Mb/s*
Minimaalne latentsus: *120 ìs*
Kättesaadavus: *mitme-tootja-tehnoloogia*
Kasutatav port või siin: *PCI*
Võrgustruktuur: *switched hubs*

ATM *(Asynchronous Transfer Mode)* on odavam kui HiPPI ja kiirem kui Fast Ethernet ning teda saab kasutada üle väga pikkade vahemaade, mis on oluline telefonifirmade jaoks. ATM protokoll pakub väiksemate lisakuludega *(overhead)* tarkvaralist liidest ning edastab efektiivsemalt lühikesi ja reaalajasõnumeid (nt. digitaalne audio ja video). Samuti on see üks suurima läbilaskevõimega võrke, mida Linux parajasti toetab. Kahjuks aga on ATM küllaltki kallis.

CAPERS

Linux' i toetus*AFAPI library*
Maksimaalne läbilaskevõime: *1,2 Mb/s*
Minimaalne latentsus: *3 ìs*
Kättesaadavus: *üldkasutatav*
Kasutatav port või siin: *SPP*
Võrgustruktuur: *kaabel kahe masina vahel*

CAPERS *(Cable Adapter for Parallel Execution and Rapid Synchronisation)* on Purdue Ülikooli PAPERS projektist väljaarenenud tehnoloogia. Sisuliselt defineerib see tarkvaralise protokolli, mis kasutab tavalist *"LapLink" SPP-to-SPP* kaablit ja realiseerib PAPERS teegi kahe Linux' i PC jaoks. Ei ole skaleeruv, kuid see-eest ei maksa peaaegu midagi.

Ethernet

Linux'i toetus: *kernel drivers*
Maksimaalne läbilaskevõime: *10 Mb/s*
Minimaalne latentsus: *100 µs*
Kättesaadavus: *üldkasutatav*
Kasutatav port või siin: *PCI*
Võrgustruktuur: *switched or unswitched hubs, or hubless bus*

Väikese koormusega võrke saab organiseerida ilma *hub*'ita siinina. Selline konfiguratsioon võimaldab minimaalsete rahaliste kuludega võrku ühendada kuni 200 masinat, kuid ei ole sobiv paralleelarvutuste jaoks. *Unswitched hub*'i lisamine jõudlust oluliselt ei paranda, kuid *switched hub*'ide kasutamisega saab juba tagada maksimaalse läbilaskvuse mitmele ühendusele samaaegselt. Linux'i toetus sellele tehnoloogiale väga laialdane, kuid tuleb silmas pidada, et variatsioonid liidese riistvaras võivad oluliselt mõjutada jõudlust, kui ei valita sobivaid draivereid. Lisaks on võimalik jõudlust parandada kanalite sidumisega (nagu kirjeldatud eelnevates peatükkides).

Fast Ethernet

Linux'i toetus: *kernel drivers*
Maksimaalne läbilaskevõime: *100 Mb/s*
Minimaalne latentsus: *80 µs*
Kättesaadavus: *üldkasutatav*
Kasutatav port või siin: *PCI*
Võrgustruktuur: *switched or unswitched hubs*

Kui võrgu nimes sisaldub Ethernet, siis on see üldjuhul mõeldud masstootmiseks ja seetõttu omab ka suhteliselt soodsat hinda. Probleem: kui ribalaiust 100 Mb/s jagab suur hulk masinaid, siis keskmine jõudlus võib olla isegi madalam kui 10 Mb/s *switched* Ethernet'i puhul. Aga vastavalt 100 Mb/s *switch*'ide hinna langemisele on neid järjest mõistlikum kasutada ka Fast Ethernet'i korral. Põhjus, miks näiteks ATM *switch*'id on nii kallid, seisneb vajaduses kommuteerida iga suhteliselt lühikest ATM *cell*'i. Mõned Fast Ethernet'*switch*'id on valmistatud eeldusega, et kommuteerimissagedus ei ole väga suur. Sellest tulenevalt võib viide liikumisel läbi *switch*'i olla küll väike, kuid liikumistee muutmine (kommuteerimine) võtta mitmeid millisekundeid. Suuremat kommuteerimissagedust nõudvate rakenduste puhul tuleb selliseid *switch*'e vältida.

Analoogselt Ethernet'iga on Fast Ethernet'i jõudluse tõstmiseks võimalik kasutada kanalite sidumist (st. vastavad draiverid on kättesaadavad).

Gigabit Ethernet

Linux' i toetus*kernel drivers*
Maksimaalne läbilaskevõime: *1000 Mb/s*
Minimaalne latentsus: *300 ìs?*
Kättesaadavus: *mitme-tootja-tehnoloogia*
Kasutatav port või siin: *PCI*
Võrgustruktuur: *switched hubs or FDR' s (Full Duplex Repeaters)*

Ka Gigabit Ethernet on mõeldud odavaks masstooteks, kuigi hinnataseme sobivale tasemele langemine võib veel veidi aega võtta. Erinevalt teistest Ethernet' idest pakub Gigabit Ethernet võimalust andmevoo juhtimiseks *(flow control)*, mis peaks tema usaldusväärsust veidi tõstma. *FDR*' id on kommutaatorid, mis jõudluse suurendamiseks kasutavad puhverdamist ja kohalikku voojuhtimist. Draiverite seisukohalt muutub väga kiirete võrkude korral kriitiliseks faktoriks PCI siini efektiivne kasutamine.

FC *(Fibre Channel)*

Linux' i toetus*puudub*
Maksimaalne läbilaskevõime: *1062 Mb/s*
Minimaalne latentsus: *?*
Kättesaadavus: *mitme-tootja-tehnoloogia*
Kasutatav port või siin: *PCI?*
Võrgustruktuur: *?*

FC eesmärk on pakkuda kõrge jõudlusega sisend- / väljundkanalit (FC kaadri kasuliku info väli *(payload)* on 2048 baiti) eelkõige ketaste jagamiseks ja teiste salvestusseadmete tarvis, mida saab FC külge ühendada otse, mitte läbi arvuti. Kui FC' st saab SCSI asendaja kõrgema nõudlusega rakendustes, siis on lootust ka hindade langemisele, aga esialgu on ta kallis ja ilma Linux' i toeta.

FireWire (IEEE 1394)

Linux' i toetus*puudub*
Maksimaalne läbilaskevõime: *196 Mb/s, 393 Mb/s*
Minimaalne latentsus: *?*
Kättesaadavus: *mitme-tootja-tehnoloogia*
Kasutatav port või siin: *PCI*
Võrgustruktuur: *random without cycles (self-configuring)*

FireWire sihiks on olla madala maksumuse ja kõrge kiirusega digitaalne võrk koduelektroonikale, näiteks digitaalse videokaamera ühendamiseks arvutiga, aga ka SCSI asendamiseks või isegi kodukino komponentide omavaheliseks ühendamiseks. FireWire võimaldab kokku ühendada kuni 64K seadet, kasutades siine ja sildu, mis ei tohi moodustada tsükleid. Konfiguratsiooni muutmine toimub automaatselt pärast seadme lisamist või eemaldamist. Kasutada saab lühikesi neljabaidiseid väikese latentsusega sõnumeid, samuti ATM sarnast isokroonset ülekannet (et hoida multimeediasõnumeid sünkroonsetena).

Kuigi FireWire ei ole suurima läbilaskvusega võrk, on väikese latentsuse võimaldamine ja suunatus tavatarbijatele (mis peaks hinnad väga madalale viima)

tegurid, mis lubavad oletada, et sellest tehnoloogiast võib lähiajal saada üks parimaid Linux PC klastrite jaoks.

HiPPI ja Serial HiPPI

Linux' i toetus*puudub*
Maksimaalne läbilaskevõime: *1600 Mb/s (serial 1200 Mb/s)*
Minimaalne latentsus: *?*
Kättesaadavus: *mitme-tootja-tehnoloogia*
Kasutatav port või siin: *EISA, PCI*
Võrgustruktuur: *switched hubs*

HiPPI *(High Performance Parallel Interface)* loodi eesmärgiga pakkuda väga suurt läbilaskevõimet superarvuti ja mingi teise seadme (superarvuti, kettamassiivi vms.) vahelise andmevahetuse jaoks ning on saanud domineerivaks standardiks superarvutitele. Serial HiPPI (mis on küll sisuliselt vastukäiv termin) on samuti populaarsust kogumas, kuna kasutab kiudoptilist kaablit 32 biti laiuse standardse HiPPI kaabli asemel. Hinnad küll aegamööda langevad, kuid on siiski veel kõrged. Seda eriti Serial HiPPI puhul, mis samas omab suuremat toetust PCI kaartide poolt kui standartne HiPPI. Linux' i toe puudumine on samuti tõsine probleem.

IrDA *(Infrared Data Association)*

Linux' i toetus*puudub?*
Maksimaalne läbilaskevõime: *1,15 Mb/s, 4 Mb/s*
Minimaalne latentsus: *?*
Kättesaadavus: *mitme-tootja-tehnoloogia*
Kasutatav port või siin: *IrDA*
Võrgustruktuur: *thin air ;-)*

IrDA on väike infrapunaseade, mis leidub näiteks paljudel sülearvutitel. Selle liidese abil on äärmiselt raske ühendada rohkem kui kahte arvutit, mistõttu on vähetõenäoline, et IrDA' t hakatakse kasutama klastrites (kuigi mõningaid katsetusi ja uurimusi on tehtud).

Myrinet

Linux' i toetus*library*
Maksimaalne läbilaskevõime: *1280 Mb/s*
Minimaalne latentsus: *9 ìs*
Kättesaadavus: *ühe-tootja-tehnoloogia*
Kasutatav port või siin: *PCI*
Võrgustruktuur: *switched hubs*

Myrinet' ist on olemas mitu varianti*LAN* versioon ja *SAN* versioon *(System Area Network*, st. võrk hulga masinate vahel, mis moodustavad suure paralleelse süsteemi), mis kasutavad erinevaid füüsilisi edastuskeskkondi ja mille karakteristikud ei ole täpselt ühesugused. Klastrite jaoks sobib paremini *SAN* versioon.

Struktuuri poolest ei paista Myrinet eriti millegagi silma, aga ta on väga efektiivselt realiseeritud. Linux' i draiverid töötavad väga hästi, kuigi esineb ka äärmiselt suuri

jõudluse kõikumisi erinevate PCI siinide realisatsioonide korral. Praeguse seisuga on Myrinet üks sobivamaid võrke klastritele, mille eelarve ei ole tõsiselt piiratud.

Parastation

Linux' i toetus:*HAL or socket library*
Maksimaalne läbilaskevõime: *125 Mb/s*
Minimaalne latentsus: *2 ìs*
Kättesaadavus: *ühe-tootja-tehnoloogia*
Kasutatav port või siin: *PCI*
Võrgustruktuur: *hubless mesh*

ParaStation on Karlsruhe Ülikooli projekt eesmärgiga luua *PVM'* i *(Parallel Virtual Machine*, tutvustus järgnevates peatükkides*)* toetav väikese latentsusega võrk. Alguses konstrueerisid nad kahe protsessoriga *ParaPC* prototüübi kasutades modifitseeritud EISA kaardil põhinevat liidest ja *BSD UNIX'* it jooksutavaid PC' sid, hiljem ehitasid juba suuremaid klastreid *DEC Alpha'* dest. Seejärel hakati tootma ka ParaStation' it toetavaid PCI kaarte. Parastation' i riistvara realiseerib kiire ja usaldusväärse sõnumiedastuse ning lihtsa tõkkesünkronisatsiooni *(barrier synchronisation)*.

PLIP

Linux' i toetus*kernel driver*
Maksimaalne läbilaskevõime: *1,2 Mb/s*
Minimaalne latentsus: *1000 ìs?*
Kättesaadavus: *üldkasutatav*
Kasutatav port või siin: *SPP*
Võrgustruktuur: *kaabel kahe masina vahel*

PLIP *(Parallel Line Interface Protocol)* võimaldab kahel Linux' i masinal omavahel suhelda läbi standardsete paralleelportide kasutades standardset *socket'* itel põhinevat tarkvara. Läbilaskvuse, latentsuse ja laiendatavuse seisukohalt ei ole see eriti tõsine võrgutehnoloogia, aga odavus (ainult ühe *"LapLink"* kaabli hind) ning ühilduvus tarkvaraga on igal juhul head omadused.

SCI

Linux' i toetus*puudub*
Maksimaalne läbilaskevõime: *4000 Mb/s*
Minimaalne latentsus: *2.7 ìs*
Kättesaadavus: *mitme-tootja-tehnoloogia*
Kasutatav port või siin: *PCI või spetsiaallahendused*
Võrgustruktuur: *?*

SCI *(Scalable Coherent Interface)* eesmärk on pakkuda suure jõudlusega vahendit, mis toetab koherentset mälupöördust üle suure arvu omavahel mälu jagavate masinate, samuti mitut tüüpi sõnumiedastust. Nii läbilaskvus kui latentsus on teiste võrgutehnoloogiatega võrreldes aukartust äratavad. Probleem aga seisneb selles, et SCI on kallis ja puudub Linux' i toetus.

SCI' d kasutatakse peamiselt erinevates loogiliselt ühtse kuid füüsiliselt jagatud mäluga masinates nagu *HP/Convex Exemplar SPP* ja *Sequent NUMA-Q 2000*. Siiski on saadaval ka SCI' d toetavad PCI kaardid ja *switch*' id.

SCSI

Linux' i toetus*kernel drivers*
Maksimaalne läbilaskevõime: *5..20 Mb/s*
Minimaalne latentsus: *?*
Kättesaadavus: *mitme-tootja-tehnoloogia*
Kasutatav port või siin: *PCI, EISA, ISA*
Võrgustruktuur: *inter-machine bus sharing SCSI devices*

SCSI *(Small Computer Systems Interconnect)* on sisend- / väljundsiin, mida kasutatakse kettaseadmete, *CD - ROM*' ide, skännerite jms. arvutiga ühendamiseks. On olemas kolm eraldi standardit: SCSI - 1, SCSI - 2 ja SCSI - 3; kiirused *Fast* ja *Ultra;* ning andmekanali laiused 8, 16 ja 32 bitti (samuti märgitakse SCSI - 3 standardis ühilduvust FireWire' ga). See kõik on küllaltki segadusttekitav, aga üldine reegel on, et hea SCSI on mõnevõrra kiirem kui EIDE ja võimaldab ühendada rohkem seadmeid ning suurema efektiivsusega.

Suhteliselt vähetuntud on küllaltki lihtne võimalus panna kaks arvutit jagama ühte SCSI siini. Selline konfiguratsioon on väga kasulik ketaste jagamiseks masinate vahel ning rikkekindluse tõstmiseks (kus üks masin võtab teise riknedes näiteks andmebaasi serveerimise enda peale). Laiendamisvõimaluse puudumine aga muudab jagatud SCSI paralleelarvutuste jaoks enamasti kasutuks.

ServerNet

Linux' i toetus*puudub*
Maksimaalne läbilaskevõime: *400 Mb/s*
Minimaalne latentsus: *3 ìs*
Kättesaadavus: *ühe-tootja-tehnoloogia*
Kasutatav port või siin: *PCI*
Võrgustruktuur: *hexagonal tree / tetrahedral lattice of hubs*

ServerNet on suure jõudlusega võrguriistvara firmalt Tandem (mis kuulub Compaq' ile ja see omakorda ühines HP' ga). "Reaalaja" tehingutöötlussüsteemides *(OLTP – Online Transaction Processing)* on Tandem tuntud kui üks juhtivaid kõrge töökindluse ja usaldatavusega süsteemide tootjaid. Ka oma ServerNet' i iseloomustavad nad lisaks suurele jõudlusele kõrge usaldusväärsuse ja andmete terviklikkusega *(data integrity)*. Veel üks huvitav omadus, mis ServerNet' il peaks olema, on andmevahetusvõime otse suvaliste seadmete vahel (st. mitte ainult protsessorite, aga ka kettaseadmete jms. vahel).

SHRIMP

Linux' i toetus*user-level memory mapped interface*
Maksimaalne läbilaskevõime: *180 Mb/s*
Minimaalne latentsus: *5 ìs*
Kättesaadavus: *research prototype*
Kasutatav port või siin: *EISA*
Võrgustruktuur: *mesh backplane*

SHRIMP *(Scalable, High-Performance, Really Inexpensive Multi-Processor)* on Princeton' i ülikooli projekt eesmärgiga ehitada Linux' it jooksutavatest PC' dest paralleelarvuti. Tõsist tööd on tehtud vähese ballastiga *(low-overhead)* suhtlus-mudelit – *"virtual memory mapped communication"* – toetava riist- ning tarkvara loomisel.

SLIP

Linux' i toetus*kernel drivers*
Maksimaalne läbilaskevõime: *0,1 Mb/s*
Minimaalne latentsus: *1000 ìs?*
Kättesaadavus: *üldkasutatav*
Kasutatav port või siin: *RS232C*
Võrgustruktuur: *kaabel kahe masina vahel*

SLIP *(Serial Line Interface Protocol)* on äärmiselt väikese jõudlusega, kuid võimaldab kahel masinal läbi tavalise RS232 pordi teostada *socket*' itel põhinevat suhtlust. Sama võimaldavad ka CSLIP *(Compressed SLIP)* ja PPP *(Point-to-Point Protocol)*. Portide ühendamiseks piisab nullmodemist. Ühenduse võib luua isegi *dial-up*' iga läbi tavalise modemi. Igal juhul latentsus on suur ja läbilaskevõime väike, seega SLIP' i kasutamisel on mõtet ainult siis, kui kuidagi teisiti enam ei saa. Märkimist väärib siiski, et kui arvutitel on kaks RS232 porti (ja paljudel on), siis põhimõtteliselt saab neist arvutitest moodustada võrgu, ühendades nad kas lineaarselt või ringina. Sellisele süsteemile on olemas isegi koormuse jaotamise *(load sharing)* tarkvara nimega EQL.

TTL_PAPERS

Linux' i toetus:*AFAPI library*
Maksimaalne läbilaskevõime: *1,6 Mb/s*
Minimaalne latentsus: *3 ìs*
Kättesaadavus: *ühe-tootja-tehnoloogia + vabakasutuses*
Kasutatav port või siin: *SPP*
Võrgustruktuur: *tree of hubs*

PAPERS *(Purdue' s Adapter for Parallel Execution and Rapid Synchronisation)* on Purdue Ülikooli projekt eesmärgiga luua skaleeritav, väikese latentsusega ning agregaatfunktsioonidega riist- ning tarkvara, mis võimaldab ehitada tavalistest PC' dest / tööjaamadest paralleelarvuteid. Agregaatfunktsioon *(aggregate function)* on funktsioon, mis töötleb korraga tervet andmehulka, mitte selle hulga üksikut elementi.

PAPERS riistvara erinevaid variante on loodud üle kümne. Laias laastus võib ära märkida kaks arendussuunda. Versioonid nimega "PAPERS" on eesmärgiks võtnud suurema jõudluse, kasutades kõikvõimalikke sobivaid tehnoloogiaid. Nimetust "TTL_PAPERS" kandvate versioonite põhirõhk on kergel reprodutseeritavusel ka väljaspool Purdue Ülikooli ning nad on seetõttu märkimisväärselt lihtsad vabakasutuses olevad lahendused, mida on võimalik realiseerida tavalisel TTL loogikal. Üks selline disain on läinud ka kommertstootmisse.

Erinevalt enamuse teiste ülikoolide riistvaralahendustest on TTL_PAPERS saavutanud laia leviku üle maailma (eelkõige ülikoolides), sest kuigi läbilaskvus on küll oluliselt piiratud, realiseerib ta väga väikese latentsusega agregaatfunktsioonsuhtluse. Isegi kiireimad sõnumivahetusele orienteeritud süsteemid ei paku võrreldavat jõudlust selliste funktsioonide täitmisel. Sellest tulenevalt sobib PAPERS eriti hästi näiteks videoseina ekraanide sünkroniseerimiseks, kõrge läbilaskvusega võrgule ligipääsude jagamiseks / organiseerimiseks, geneetiliste algoritmide tulemuste võrdlemiseks globaalse optimumi otsimisel jne.

USB *(Universal Serial Bus)*

Linux' i toetus*kernel driver*
Maksimaalne läbilaskevõime: *12 Mb/s*
Minimaalne latentsus: *?*
Kättesaadavus: *üldkasutatav*
Kasutatav port või siin: *USB*
Võrgustruktuur: *bus*

USB võimaldab ühtsele siinile lisada kuni 127 välisseadet klaviatuuridest kuni videokonverentsi kaamerateni. Seadmete lisamiseks ei ole neid tarvis välja lülitada *(hot-pluggable)*. Arvutite omavahelisest ühendamisest USB abil ei ole veel eriti palju räägitud. Igal juhul on USB muutumas standardseks pordiks emaplaatidel ja väärib põhjalikumat uurimist (eriti senikaua, kuni FireWire veel piisavalt levinud ei ole).

WAPERS

Linux' i toetus*AFAPI library*
Maksimaalne läbilaskevõime: *0,4 Mb/s*
Minimaalne latentsus: *3 ìs*
Kättesaadavus: *vabakasutuses*
Kasutatav port või siin: *SPP*
Võrgustruktuur: *wiring pattern between 2 - 64 machines*

WAPERS *(Wired-AND Adapter for Parallel Execution and Rapid Synchronisation)* on välja kasvanud Purdue Ülikooli PAPERS projektist. Korrektse realisatsiooni korral on SPP' l neli bitti avatud kollektoriga väljundeid, mida on võimalik teiste arvutite samade väljaviikudega kokku ühendada moodustamaks 4 biti laiust *wired-AND*' i. Selline lahendus on elektriliselt tundlik ja ühendatavate arvutite maksimaalne arv sõltub portide analoogomadustest. Tavaliselt piirdub masinate arv 7..8-ga. Kuigi hind ja latentsus on väga väikesed, on seda ka läbilaskvus. Seetõttu sobib WAPERS eelkõige klastri sekundaarseks võrguks agregaatfunktsioonide jaoks.

Võrguriistvaraga suhtlemise põhimõtted

Tarkvara ja võrguriistvara vahelise suhtlemise põhilisi meetodeid on kolm: *socket*'id, seadmedraiverid ja kasutajateegid *(user-level libraries)*.

Socket'id on levinuim madaltaseme võrguliides (tarkvaraline). Suurem osa standardsest võrguriistvarast toetab vähemalt kahte liiki *socket* protokolle: UDP ja TCP. Mõlemad võimaldavad saata suvalise suurusega andmeplokke ühest masinast teise ning mõlemad põhjustavad võrgu suurt latentsust.

UDP protokoll *(User Datagram Protocol*, ka *Unreliable Datagram Processing :)* võimaldab andmeplokki saata individuaalse sõnumina, mis aga võib edastuse käigus kaduma minna või jõuda sihtpunkti mitmes eksemplaris. Mitme sõnumi saatmise korral võib sõnumite kohalejõudmise järjekord erineda saatmisjärjestusest. UDP sõnumi saatja ei saa automaatselt kviteeringut *(acknowledgment)*, selle saatmise peab kasutaja ise oma koodi kirjutama, kui tal seda vaja on. Küll aga tagab UDP sõnumi sisu muutumatuse – kui sõnum kohale jõuab, siis on tema sisu korrektne. UDP positiivseteks omadusteks on tema kiirus teiste *socket* protokollidega võrreldes ning ühenduseta *(connectionless)* edastus – iga sõnum on teistest sõltumatu. UDP'd võib võrrelda kirja saatmisega: samale aadressile võib saata mitu kirja, kuid kõik nad on üksteisest sõltumatud; ning inimeste arv, kellele kirju saab saata, pole samuti piiratud.

TCP *(Transmission Control Protocol)* on usaldusväärne ühenduspõhine protokoll. Saadetavaid andmeplokke ei vaadelda eraldi sõnumitena, vaid kui pidevat infovoogu saatjalt vastuvõtjale. See erineb oluliselt UDP'st – nüüd on iga plokk lihtsalt osake andmevoost ja plokkide eraldamine baidivoost on kasutaja enda probleem. Ühendused on võrguprobleemide suhtes tundlikumad ja ühenduste arv on piiratud. Usaldusväärsuse kasv tuleb teenindusinfo arvelt, mis kasutaja jaoks on ballast *(overhead)*. Üheks positiivseks omaduseks (eriti just klastrite seisukohalt) on TCP suutlikkus pakkida väikseid sõnumeid kokku suuremateks, võrguriistvaraga paremini sobivateks pakettideks. Lühikestest mittetavapärase suurusega sõnumitest koosnevate gruppide korral võib TCP põhimõtteliselt saavutada parema jõudluse kui UDP. Teiseks on olemas võimalus ehitada võrk, kus arvutid on ühendatud usaldusväärsete füüsiliste otseühendustega, ning sellel TCP'd simuleerides saavutada lihtsate vahenditega efektiivne tulemus – võrgu jõudlus on palju parem kui tõelise TCP'ga, aga tarkvara / koodi semantika jääb praktiliselt samasuguseks.

Järgmine meetod on otsene seadmedraiverite kasutamine. UDP ja TCP sisaldavad suurel hulgal *socket*'ite haldamiseks vajalikku teenindusinfot (mis näiteks võimaldab mitmel TCP ühendusel jagada ühte füüsilist võrguliidest). Seadmedraiver aga realiseerib ainult mõned lihtsad transpordifunktsioonid info saatmiseks läbi konkreetse riistvaralise seadme (stiilis *open(), read(), write()*), mille tulemusena liidese kiirus kasvab märgatavalt.

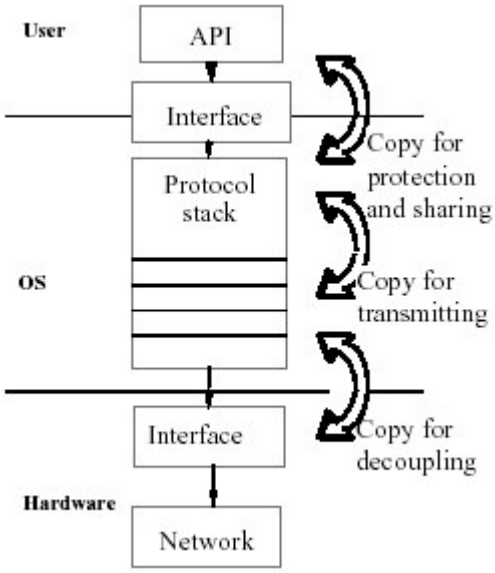
User-level library suhtleb otse seadme registritega. See on asi, mida operatsioonisüsteemi (OS) seisukohalt ei tohiks mingil juhul teha. OS'i üks peamisi ülesandeid ongi ju ise juhtida / kontrollida juurdepääsu riistvarale. Samas põhjustab OS'i funktsioonide väljakutsumine tõsise ajalise viite vähemalt kümnetes mikrosekundites, mis selliste võrgutehnoloogiate nagu näiteks TTL_PAPERS (latentsus ainult 3 µs) korral on lubamatu. Ainus viis vältida seda ajakulu on kasutajakoodi otsene

pöördumine riistvara poole. Põhiküsimuseks on, kuidas samal ajal mitte rikkuda OS' i juhtimisõigusi (vastasel korral ta lihtsalt ei lase kasutajaprogrammil töötada). Selleks on mitmeid erinevaid meetodeid sõltuvalt OS' ist (nt *memory mapping* või *ioperm()* kasutamine).

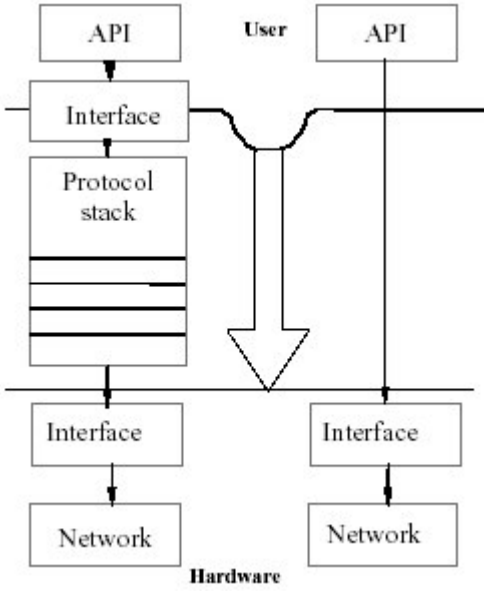
Ajakulud tekivad OS' is järgnevatel põhjustel:

- Protokolli *stacki* funktsioonid:
  - Andmete terviklikkuse tagamine
  - Kasutaja protsesside kaitse
  - Ressursside jagamine
- Kasutatavad mehhanismid:
  - Andmete kopeerimine
  - CRC kontrollimine
  - Voojuhtimine
  - Multipleksimine ja demultipleksimine
  - Süsteemikutsed *(system calls)*

Otse riistvaraga suheldes peab muidugi meeles pidama, et sel juhul need teenused puuduvad, ning vastavate tagajärgedega arvestama.



J15. Ajakulu OS' i läbimisel.



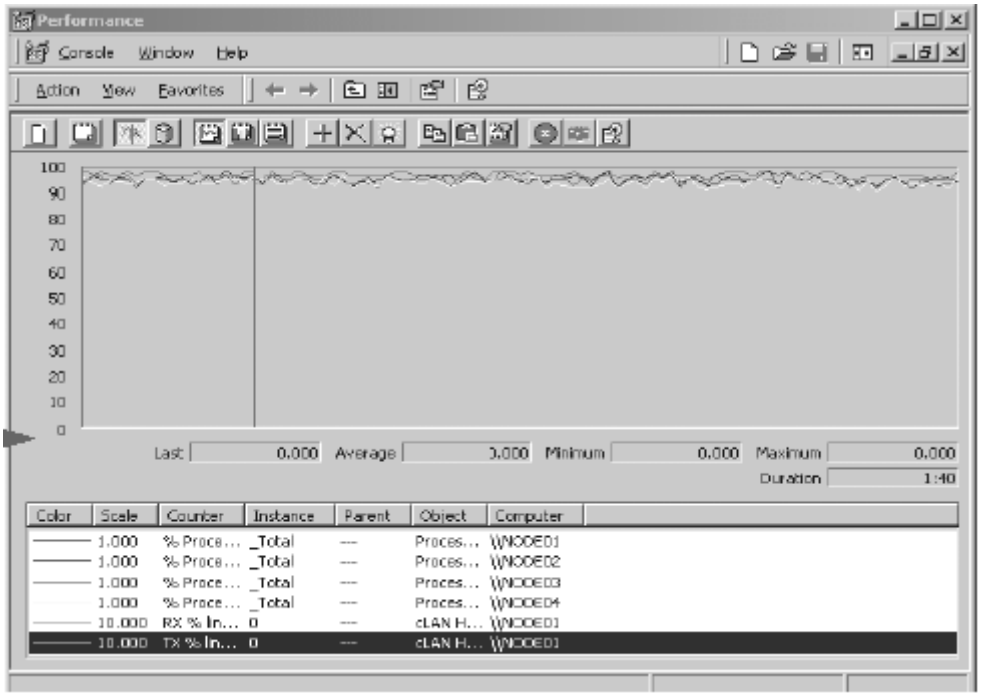
J16. Kasutaja otsene pöördumine riistavara poole.

Tehnoloogiate võrdlusi

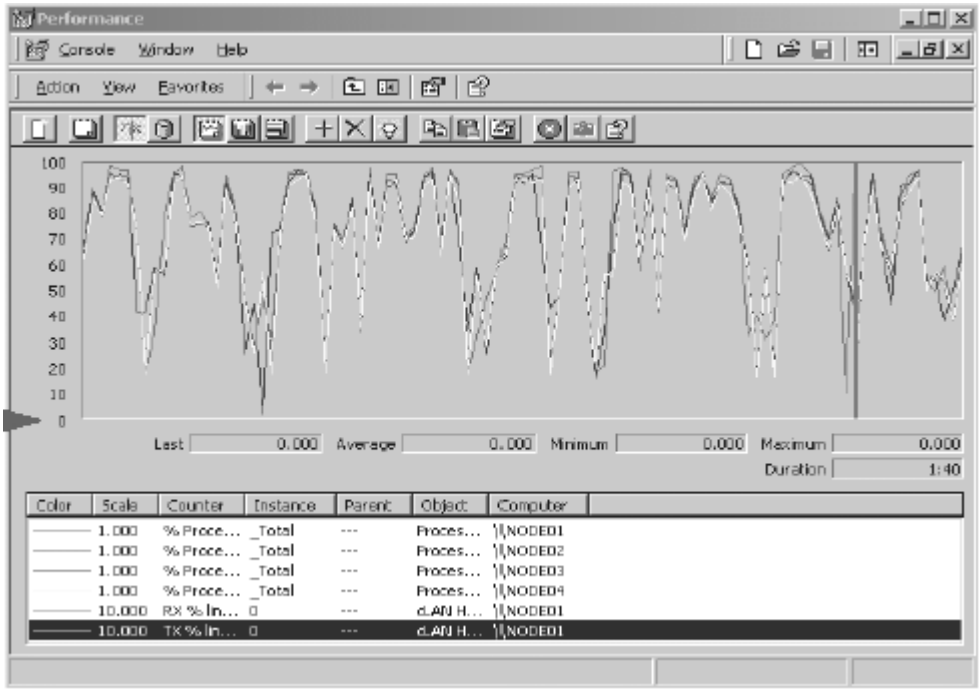
Illustreerimaks kasutatavate tehnoloogiate mõju klastri töövõimele on järgnevas toodud paar näidet. Kuna eesmärgiks on lihtsalt põhiprobleemide tutvustamine, ei ole konkreetsete katsete detaile kirjeldatud.

Nagu eelnevates peatükkides juba mainiti, on klastrite korral äärmiselt oluline mitte ainult võrgu läbilaskvus, vaid ka latentsus. Antud näites on jooksutatud viidete suhtes tundlikku koodi, mis tegeleb maatriksarvutustega ja saadab selle käigus suurel hulgal sõnumeid arvutikobara masinate vahel. Kui protsessor peab andme-ülekande viivituste tõttu ootama, siis selle aja jooksul ta tööd ei tee ja jõudlus langeb.

Vaadeldud on kahte tehnoloogiat – ühel juhul on ühenduskanaliks 100BaseT *switched* Ethernet, teisel juhul Emulex koos VIA protokolliga *(Virtual Interface Architecture*, eesmärk on pakkuda suurt läbilaskevõimet ja väikest latentsust, kasutades OS' ist möödaminekut ning enamasti ka modifitseeritud riistvara.). Mõlemal juhul on mõõdetud protsessorite tööga koormatust. Emulex + VIA korral on protsessorid küllaltki ühtlaselt koormatud (J17), aga Ethernet' i kasutades peavad protsessorid tihti tühjalt ootama (J18) ning seetõttu on klastri jõudlus väiksem. Samas on oluliselt madalam ka Ethernet' i rahaline maksumus.



J17. Protsessorite koormatus: Emulex + VIA.



J18. Protsessorite koormatus: Ethernet.

Joonistel J19 ja J20 on võrreldud nelja levinud võrku – Myrinet, Emulex, Gigabit Ethernet ja Fast Ethernet – latentsuse ja läbilaskvuse seisukohalt.

J19. Latentsuse võrdlus.

J20. Läbilaskvuse võrdlus.

Operatsioonisüsteemid

Operatsioonisüsteem on klastri oluline komponent. Kui varasematel OS' idel puudus igasugune paralleelsuse tugi, siis tänapäevastel võib seda juba suuremal või vähemal määral täheldada. Järgnevalt on toodud laialt levinud näide "*The Parallel Computing Store"*, mis ühest küljest iseloomustab operatsioonisüsteeme paralleelsuse seisukohalt, teisalt aga on kasutatav ka üldisemal juhul – paralleel-arvutuste olemuse selgitamiseks.

"The Parallel Computing Store"

Vaatleme suurt kauplust 8 kassaga, mis asuvad üheskoos poe esiosas. Olgu iga kassa / kassapidaja protsessori ja klient arvutiprogrammi analoog. Kliendi ostude hulk on samaväärne programmi suurusega (töömahuga).

Single-tasking OS

Avatud on üks kassa ning see teenindab kliente ühekaupa (st. üks klient korraga).
Näide: MS DOS.

Multitasking OS

Avatud on üks kassa, aga nüüd töötleb ta ainult teatava osa kliendi ostudest, võtab ette järgmise inimese ja töötleb osa tema ostukorvist. Jääb mulje, et kõik liiguvad koos läbi kassa, aga kui järjekorras oleks ainult üks inimene, saaks ta valmis kiiremini.
Näide: UNIX, NT ühel protsessoril.

Multitasking OS with Multiple CPU's

Nüüd on poes avatud mitu kassat. Iga kassa teenindab ühte klienti korraga ning järjekord liigub edasi palju kiiremini kui enne. Seda nimetatakse SMP *(Symmetric Multi-Processing*). Kuigi avatud on rohkem kassasid, ei saa konkreetse ostja teenindamine olla kiirem kui üheainsa kliendi ja ühe kassa korral.
Näide: UNIX või NT mitme protsessoriga (samal emaplaadil).

Threads on a Multitasking OS with Multiple CPU's

Kui jaotada kliendi tellimus (ostukorv) osadeks, on võimalus kasutada sama tellimuse töötlemiseks mitut kassat korraga ja saada teenendatud kiiremini. Kõigepealt peaks eeldama, et kliendil on palju kaupu, sest tellimuse osadeks jaotamine võtab oma aja ning see peab saama kompenseeritud mitme kassa kasutamisest tuleneva kiiruse kasvuga. Teoreetiliselt peaks olema võimalik saada teenendatud *n* korda kiiremini*, kus *n* on kassade arv. Lõpparve arvutamiseks peavad kassapidajad omavahel vestlema või jälgima eemalt teiste kassade arveid, või isegi ümber teiste kassade ringi hiilima, et endale vajalikku infot kätte saada. Efektiivselt ühte poe osasse kuhjatud kassade arv on siiski piiratud.

* Kiirenemist piirab ka Amdal' i seadus – teenendamine ei saa olla kiirem, kui on vajalik programmi kõige aeglasema (suurema) mittejagatava osa töötlemiseks.

Näide: UNIX või NT mitme protsessoriga (samal emaplaadil), kusjuures jooksu-tatakse *multi-threaded* programme.

Sending Messages on Multitasking OS with Multiple CPU's

Jõudluse suurendamiseks lisab pood müügisaali teise otsa veel 8 kassat. Kuna uued kassad on vanadest kaugel, peavad kassapidajad mitme peale ühte klienti teenendades kasutama telefone, et lõpparvet kokku panna. Tekkinud distants lisab suhtlusele kuluvat aega, aga kui suhtlemise maht minimeeritakse, ei ole see probleem. Kas kliendil on mõtet kasutada kõiki kassasid korraga, sõltub tema tellimuse suurusest ning kassade kommunikatsioonile kuluvast "ballastajast".

Näide: Üks või mitu koopiat UNIX'ist või NT'st ühel või mitmel emaplaadil + suhtlemine sõnumite abil.

Eelnevad stsenaariumid ei pruugi olla küll päris täpsed analoogiad, kuid annavad hea ettekujutuse paralleelarvutuse piirangutest – erinevalt üksikust protsessorist on oluline ka kommunikatsioon.

Operatsioonisüsteemi ülesanded

Kui minna konkreetsemaks ja vaadelda tegelikke operatsioonisüsteeme, mitte kauplusi, siis üks kaasaegne OS peab kasutajale pakkuma kahte peamist teenust:

- Looma virtuaalmasina – peitma kasutaja eest igasugused riistvara spetsiifilised omadused, mis erinevatel komponentidel võivad olla erinevad, aga kasutaja jaoks on mitteolulised.
- Jagama riistvaralisi ressursse kasutajate või töös olevate protsesside vahel.

Need teenused täidetakse järgmiste operatsioonisüsteemi funktsioonide abil:

- Protsessihaldus – OS peab protsessidele ressursse eraldama, võimaldama protsessidel jagada ja vahetada andmeid, kaitsma protsessi ressursse teiste protsesside eest, võimaldama protsesside omavahelist sünkroniseerimist, ja protsesse käivitama, lõpetama ning ajutiseks peatama vastavalt erinevatele prioriteetidele ning planeerimisalgoritmile.
- Mäluhaldus – OS peab teostama mälulükkeid *(relocation)*, kaitset, jagamist ning käsitsema mälu loogilist ja füüsilist struktuuri.
- Failisüsteemi haldamine – OS peab haldama muuhulgas failide struktuuri, organisatsiooni, nimetamist, juurdepääsu, kasutamist ja lukustamist.
- Sisend / väljund – OS peab suhtlema paljude erinevate s/v-seadmetega, mis jagunevad laias laastus kolmeks:
  - Inimbetavad (nt. monitorid, printerid, klaviatuurid, hiired, kõlarid).
  - Masinbetavad (nt. kettaseadmed, digikaamerad jms.).
  - Kommunikatsioonriistvara (nt. modemid, võrgukaardid).

Paralleelsetes süsteemides on eelnimetatud ülesanded keerulisemad ning lisandub ka uusi vajalikke funktsioone. Tänapäeval peab OS'i puuduste kõrvaldamiseks kasutama vahetarkvara *(middleware)*, aga aegamööda lisandub ka uutele OS'idele paralleeltöötlust toetavaid funktsioone. Peamised teemad, mida

järgmise generatsiooni operatsioonisüsteemid klastrite toetamiseks silmas peaks pidama, on:

- Rikete haldus – OS peaks mõne komponendi rikke korral tema funktsioonid võimalusel teistele komponentidel andma, nii et rakendus probleemi ei märka.
- Koormuse jaotamine efektiivselt kasutadaolevate masinate vahel, oskus kohaneda muutuva konfiguratsiooniga (kui arvuteid lisatakse või eemaldatakse).
- Arvutuste paralleliseerimine.
  - Paralleliseeriv kompilaator – paralleelselt töötada võivad programmi osad leitakse kompileerimise ajal.
  - Paralleelsed ja klastriteadlikud rakendused – paralleelsus on programmi juba sisse kirjutatud, kas siis sõnumite saatmist kasutades (programmeerija jaoks raskem, kuid vahel efektiivsem) või *thread*'ide abil (las klaster (tema OS või vahetarkvara) ise jagab *thread*'e sobivatele masinatele).

Jagatud seadmete mudel – klastri arvutid jagavad omavahel ressursse (kõvaketast, mälu vms.). Sel juhul on vaja tagada kõikvõimalike konfliktide lahendamine: organiseerida ressurssidele juurdepääsu jagamine, sünkroniseerimine. Kannatada võib arvutikobara jõudlus, kuna seadmete jagamiseks on vaja teenindusliku lisainfo liigutamine masinate vahel, samuti peavad protsessid mõnikord ootama, kuni mingi ressurss vabaneb.

Ei-jaga-midagi mudel (Shared Nothing Model) – iga klastri sõlm omab isiklikke riistvaralisi ressursse ja mingit jagamist ei toimu. Kui ühes sõlmes tekib rike, võtab vabanevad ressursid (kui neid on; nt. kettaseadmed) üle mõni teine masin.

Levinumad operatsioonisüsteemid klastrites

Peamised klastrites kasutatavad OS'id on Linux (eriti Redhat), MS Windows (NT, 2000) ja SUN Solaris. Kuna Linux on tasuta saadaval ning areneb väga kiiresti, siis on ta küllaltki populaarne, eriti kõikvõimalike ülikoolide ja uurimisasutuste juures. Kommertssüsteemid on kallid, aga pakuvad näiteks kasutajatuge. OS'i valikul tuleb lähtuda iga konkreetse klastri eesmärkidest ning rahalistest võimalustest.

Eelnimetatud ei ole loomulikult ainsad arvutikobarates kasutatavad operatsioonisüsteemid. Näiteks on kasutatud ka Free BSD'd, MAC OS'i ja Scyld Beowulf Cluster Operating System'it (spetsiaalselt klastritele sobitatud tasuline Linux'i distributsioon. Aga enamasti ehitatakse Beowulf klastreid ikka tasuta Linux'iga.).

Vahetarkvara

Vahetarkvara on peamine komponent, mis eristab paralleelarvutusi teostava süsteemi tavalistest võrku ühendatud arvutitest. Vahetarkvara on tarkvarakiht, mis pakub ühesugust juurdepääsu klastri erinevatele arvutitele, sõltumata neil kasutatavast operatsioonisüsteemist. See kontseptsioon on analoogne näiteks Windows 2000' e HAL' iga *(Hardware Abstraction Layer)*, mis pakub ühtset ja porditavat *(portable)* juurdepääsu PC riistvarale.

Järgnevalt on toodud teenused ja funktsioonid, mida klastri vahetarkvara peaks pakkuma (sõltub muidugi rakenduse eesmärkidest):

- Üksainus sisenemispunkt – kasutaja logib ennast klastrisse, mitte konkreetsesse arvutisse.
- Üksainus failide hierarhia – kasutaja näeb ühtainust failihierarhiat ühesainsas juurkataloogis.
- Üksainus juhtimispunkt – klastrit juhib konkreetne arvuti.
- Üksainus virtuaalvõrgusüsteem – erinevate füüsiliste võrkude samaaegne kasutamine peab klastri sõlmedele olema läbipaistev. Nemad näevad ühtset loogilist võrku.
- Üksainus mäluruum – hajus jagatud mälu võimaldab programmidel muutujaid jagada.
- Üksainus tööde haldamise süsteem – kasutaja ei pea spetsifitseerima, millises võrgusõlmes tema programm tööle hakkab.
- Üksainus kasutajaliides – ühtne (graafiline) liides kõigile kasutajatele sõltumata nende klastrisse sisenemise asukohast (nt. välisvõrgust).
- Üksainus sisend- / väljundruum – klastri sõlm ei pea teadma s/v-seadmete tegelikku füüsilist asukohta.
- Üksainus protsessiruum – protsesside identifitseerimise skeem hõlmab tervet klastrit (unikaalsed ID' d). Protsessid võivad omavahel suhelda sõltumata füüsilisest asukohast.
- Kontrollpunktid – protsesside ja vahetulemuste perioodiline salvestamine võimaldamaks valutut taastumist riketest.
- Protsesside ränne / migratsioon – töötavaid protsesse on võimalik tõsta ühest masinast teise. Võimaldab realiseerida koormuse jaotamist.

Seega vahetarkvara peab looma ühtse süsteemi mulje *(Single System Image)*. Samuti vastutab ta klastri kõrge kättesaadavuse eest, teostades koormuse jaotamist ja reageerides komponentide riketele. Tulevikus võiks enamust neist funktsioonidest täita operatsioonisüsteem – vahetarkvara üks peamisi rolle ongi teostada neid funktsioone, mida OS võiks ise teha, aga praegu veel ei tee.

Paralleelprogrammeerimise abivahendid

Tavalistel järjestikulistel programmidel ei ole kalduvust klastrisse sattumise korral iseenesest paralleelselt tööle hakata, vaja on paralleelsus juba programmi loomisel lisada. Olemas on mitmeid *library*'sid, mis programmeerijat selle raske ülesande juures abistavad. On ka spetsiaalselt paralleelprogrammide loomiseks mõeldud keeli. Järgnevalt väike ülevaade neist.

Kommunikatsiooniteegid *(libraries) paralleelarvutustele*

Enamasti koosneb klaster teatud hulgast arvutitest, millel on igaühel isiklik (st. mitte teistega jagatud) mälu. Seega ainus võimalus masinate vaheliseks suhtlemiseks on kasutada võrku. Selleks kasutatakse sõnumite saatmist *(message passing)*. Teekide peamine ülesanne on tagada programmide porditavus – ühilduvus erinevate süsteemidega. Kaks levinumat teeki on PVM ja MPI.

PVM *(Parallel Virtual Machine)* loodi esialgselt tööjaamade võrgu jaoks, kuid nüüdseks toetab paljusid erinevaid paralleelseid süsteeme. PVM võimaldab heterogeenset arvutisüsteemi käsitleda üheainsa paralleelse virtuaalmasinana, tehes programmeerija jaoks nähtamatuks sõnumite marsruutimise, andmeteisendused ja tegumite planeerimise *(task scheduling)*. PVM'i kasutamine on küllaltki lihtne ja intuitiivne. Programmeerija loob rakenduse kui kogumi omavahel koostööd tegevaid tarkvarakomponente. PVM pakub vahendeid nende komponentide / tegumite initsialiseerimiseks ja lõpetamiseks üle võrgu, samuti tegumite vaheliseks suhtlemiseks ja sünkronisatsiooniks. PVM sisaldab ka mõningaid veakindluse tagamise vahendeid, võimaldades kirjutada rakendusi, mis suudavad üle elada tegumite või arvutite riknemise ja koormusjaotuse muutumise.

MPI *(Message Passing Interface)* on erinevalt PVM'ist standardiseeritud süsteem, mida toetavad paljud riistvaratootjad ning mis pakub rohkem funktsioone (kuid on ka selle võrra keerulisem).

Mõlemad teegid on populaarsed ja omavad pooldajaid ning vastaseid. Kuna nad ei ole funktsionaalselt täiesti kattuvad, siis valiku tegemisel tuleb lähtuda konkreetse rakenduse eesmärgist.

Rakenduste loomise paketid / keeled

Loodud on palju erinevaid paralleelprogrammeerimise keeli, kuid käesolev referaat neid detailsemalt ei vaatle. Küll aga võib välja tuua nende soovitavaid omadusi ja arengusuundi.

Paralleelprogrammeerimise mudelitel ja keeltel peaks olema järgmised omadused:

- Kerge programmeerida – programmi täpse tööstruktuuri peaks määrama translatsioonimehhanism (kompilaator ja töökeskkond *(run-time system)*), mitte programmeerija. Seega peaks mudel suutma:
  - `Dekomponeerida programmi paralleelseteks` *thread*'ideks.
  - `Jagada need` *thread*'id protsessoritele laiali.

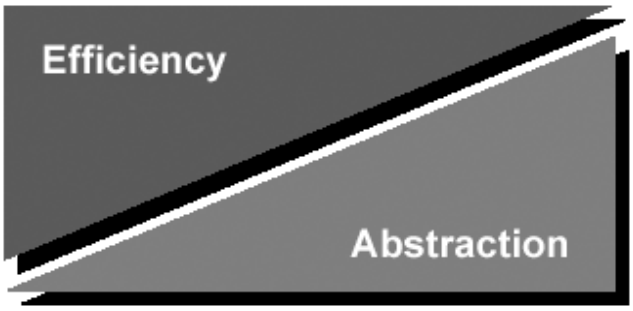
    - Organiseerida nendevahelist suhtlust.
    - Organiseerida nendevahelist sünkroniseerimist.
- Tarkvara arendusmetoodika olemasolu, et programmeerija poolt programmikoodina antud informatsioon viia kooskõlla selle programmi käivitamiseks vajaliku detailse struktuuriga.
- Sõltumatus arhitektuurist – võimaldab programme kasutada erinevates paralleelarvutites ilma vajaduseta teostada tõsisemaid muudatusi programmikoodis. Antud omadus on oluline ka seetõttu, et paljude rakendusprogrammide eluiga võib olla kümneid aastaid, aga arvuti-arhitektuuride (eriti paralleelarvutite arhitektuuride) eluiga on suhteliselt lühike.
- Kergesti arusaadav – mudel peaks olema kergesti arusaadav ja õpetatav, muidu on võimatu treenida olemasolevaid tarkvaraarendajaid seda kasutama.
- Realiseeritav kõrge efektiivsusega – programmid peaks efektiivselt töötama erinevatel paralleelarhitektuuridel.
- Pakub täpset informatsiooni programmidega kaasnevate kulutuste kohta, sealhulgas programmi tööaeg, protsessorite kasutamise efektiivsus, arendustöö maksumus.

Need kriteeriumid peegeldavad usku sellesse, et paralleelsüsteemide arenguks on vaja toimivat paralleeltarkvaratööstust, mis baseerub programmide porditavusel ja efektiivsusel.

Vastavalt Skillcorn' i jaotusele jagunevad paralleelprogrammeerimise mudelid kuude kategooriasse sõltuvalt abstraktsioonitasemest:

1. *Nothing Explicit, Parallelism Implicit* – mudelid, mis varjavad paralleelsuse täielikult, kirjeldades ainult programmi eesmärki, mitte selle saavutamise viisi.
2. *Parallelism Explicit, Decomposition Implicit* – tarkvaraarendajad on teadlikud, et saab kasutada paralleelsust, ja peavad programmikoodis selleks soovi avaldama, kuid ei tea, kuidas see paralleelsus programmi töö ajal tegelikult välja hakkab nägema.
3. *Decomposition Explicit, Mapping Implicit* – mudel nõuab programmeerijalt otsuseid töö paralleelseteks osadeks jagamise kohta, kuid vabastab ta selle tagajärgedest (osade jaotamine protsessoritele, kommunikatsioon, sünkronisatsioon).
4. *Mapping Explicit, Communication Implicit* – programmeerija peab otsustama ka programmi komponentide paigutuse protsessoritele. Nõuab enamasti eelteadmisi masinatest ja neid ühendavast võrgust ning seetõttu ei ole saadav kood eriti porditav.
5. *Communication Explicit, Synchronization Implicit* – arendaja peab ise määrama kõik, välja arvatud täpsed ajastusküsimused.
6. *Everything Explicit* – programmeerija peab määrama kõik detailid, abstraktsioon puudub (nt. PVM, MPI).

Rusikareegel ütleb, et programmi efektiivsus ja abstraktsioonitase on pöörd-võrdelises sõltuvuses:



J21. Efektiivsus vs. abstraktsioon.

Enamus uuemaid mudeleid jääb kusagile selle joonise keskele, üritades leida kompromissi väljendusrikkuse ja efektiivsuse vahel. Siiski leidub ka mudeleid, mis on abstraktsed ja efektiivselt realiseeritavad ühekorraga, avades väljavaate paral-lelismile kui osale tavalisest tarkvaratööstusest, mitte ainult kõrge jõudlusega arvutusmaailmast nagu praegu.

Millele klastri loomisel tähelepanu pöörata?

Enne kui hakata klastrit ehitama, oleks vaja olukord põhjalikult läbi mõelda (seda eriti juhul, kui ehitatakse täiesti uus klaster, mitte ei kasutada juba olemasolevat arvutite kogumit). Tihti kipuvad mõned asjad kahe silma vahele jääma, seetõttu on järgnevalt toodud mitmesuguseid soovitusi ja küsimusi, mida esitada endale ja kõigile asjaga seotud isikutele.

Ootuste tuvastamine

Projekti edu või läbikukkumist ei mõõdeta tihti mitte saavutuste, vaid ootuste täitmise järgi. Need küsimused proovivad vastavaid ootusi määrata:

- Kas klastri jõudlusnõuded on dokumenteeritud? Kui jah, siis täpsustage nende täitmise kontroll (milliste mõõtmiste abil nõuete täitmist määratakse).
- Kas klastri konfiguratsiooni on hinnatud ja verifitseeritud nõuetest lähtuvalt?
    - Rakenduse tüüp
    - Protsessor(id)
    - Mälu
    - Vahemälu *(cache)*
    - Kettaseadmed, RAID
    - Magnetlintsalvesti (varukoopiate tegemiseks)
    - Adapterid
- Kas on kindlaks määratud nõuded kettaseadmetele ja ketaste paigutus? Seda nii klastrisiseste kui -väliste salvestusseadmete jaoks.
- Kas klaster vajab kõrge kiiruse ja väikese latentsusega võrku (Myrinet)?
- Mille jaoks klastrit kasutama hakatakse?
- Millised on teie ootused sellele klastrile, kirjeldatuna kuni 10 sõnaga?
- Milline saab olema klastri nimi?
- Kas olete teadlik mistahes probleemidest, mis võivad kahjustada teie võimet see klaster valmis ehitada?

Asukohaga seotud küsimused

Neile küsimustele tuleks vastata enne klastri kokkupaneku algust.

- Kas klaster hakkab paiknema olemasolevas arvutiruumis? Kui jah, siis kas seal on piisavalt ruumi?
- Kas on olemas tõstetud põrand *(raised floor)*? Kui jah, siis kas see pakub piisavalt ruumi kaablite jaoks?
- Kas põrand kannatab klastri raskust?
- Kas on, või saab olema, piisavalt elektrivõimsust, sobiv pinge ja sobivad pistikupesad?
- Kas ventilatsioon saab lisanduva soojusega hakkama?
- Kas on teada mingeid plaanipäraseid muutusi ruumi varustatuses elektri ja muude infrastruktuuridega? Kuidas see mõjutab klastrit?

Riistvara / konfiguratsiooni küsimused

Neile küsimustele tuleks vastata enne klastri installeerimise algust.

- Milliseid lisaseadmeid klastris veel kasutatud on?
- Millised on IP aadresside vahemikud (arvutussõlmedele, juhtsõlmedele, *switch*' idele jms.)?
- Millist nimetuste skeemi kasutate (sõlmed, *rack*' id, haldus jms.)? Kas alustate nulli või ühega? Kas on mingeid nõudeid ees- või järelliite suhtes?
- Kas on konkreetseid nõudeid siltidele (analoogselt eelmisele küsimusele, aga füüsiliselt eksisteerivate siltide kohta)?
- Kas kasutate VLAN' e? Kui jah, siis mitut? Millise sektsioneerimis-põhimõttega?
- Millised on turva- ja autentimisnõuded?
- Kuidas ühendatakse välised salvestusseadmed?
- Millist tööde läbilaskvust klastrilt nõutakse?
- Kas teie organisatsiooni võrguosakonnal on teadaolevaid plaane teha muudatusi või lisandusi võrguseadmetele? Kuidas see mõjutab klastrit?
- Kas võrguosakonnal on erinõudeid ühilduvuse osas (tootjafirma, eriseadmed, vask, kiudoptika jms.)?
- Kas on olemas riistvarale vastav tarkvara ja draiverid?

Tarkvaraküsimused

Neile küsimustele tuleks samuti vastata enne klastri installeerimise algust.

- Milliseid avatud lähtekoodiga programme on plaanis installeerida?
- Milliseid kommertstooteid on kavas installeerida?
- Kas teil on juba mõni klaster, kus vastav tarkvara jookseb?
- Kas on vaja spetsiaalseid / kommertskompilaatoreid?
- Kas operatsioonisüsteem toetab kõiki planeeritud funktsioone?
- Kas mõnel vajalikul tarkvaratootel on erinõudeid teiste tarkvaratoodete versioonide suhtes?
- Kas on kindlaks määratud varukoopiate tegemise plaan? Defineerige varundamise nõuded, sagedus ja maht.

Beowulf'i ajalugu ja olemus

Selle referaadi käigus on korduvalt mainitud Beowulf tüüpi klastreid. Käesolev peatükk annab selle klastriliigi kohta täpsemat informatsiooni.

1994. aasta suvel Thomas Sterling ja Don Becker, töötades CESDIS'es *(Center of Excellence in Space Data and Information Sciences, NASA)* projekti ESS *(Earth and Space Sciences project)* raames, ehitasid 16 DX4 protsessoriga arvutist koosneva klastri, mis kasutas seotud kanalitega Ethernet'i. Nad nimetasid oma masina Beowulf'iks. Masinat saatis kohene edu ja nende idee teha *COTS (Commodity Off The Shelf)* komponentidest süsteeme spetsiifiliste arvutusprobleemide lahendamiseks levis kõigepealt NASA's ja seejärel akadeemilistes ja uurimustööga tegelevates ringkondades. Selle masina ehitamisest on välja kasvanud *Beowulf Project* ning Beowulf klastrid on saanud eraldi liigiks kõrge jõudlusega klastrite kogukonnas.

ESS projekti üks eesmärke oli kindlaks määrata paralleelarvutite kasutuskõlblikkus Maad ja kosmost uurivate teadusharude jaoks. Esimene Beowulf ehitati eesmärgiga leida lahendusi probleemidele, mis on seotud suurte andmehulkadega, sest selliseid probleeme tuleb ESS'i temaatikas tihti ette.

On tõenäoline, et see oli lihtsalt tehnika arengu seisukohalt loogiline hetk Beowulf tüüpi arvutite tekkeks. Viimaste aastakümnete jooksul on toimunud sündmused, mille tagajärgedeks on Beowulf'ile vajalikud eeldused. *COTS* tööstus toodab täiesti kasutusvalmis alamsüsteeme (mikroprotsessorid, emaplaadid, kettad, võrgukaardid). Masstootmise tekitatud konkurents on viinud hinnad alla ja töökindluse üles. Avatud lähtekoodiga tarkvara, eriti Linux OS, GNU kompilaatorid ja programmeerimisvahendid ning MPI ja PVM sõnumivahetusteegid varustavad meid riistvarast sõltumatu tarkvaraga. Järjest on kasvanud nõudmine kõrge jõudlusega arvutuste järele. Kõik see kokku viib mõttele, et Beowulf'i teke oli loomulik evolutsiooniline sündmus.

Pidevalt on juttu mikroprotsessorite jõudluse järjekordsest tõusmisest, aga Beowulf'i seisukohalt on isegi olulisem võrgutehnoloogia jõudluse ja hinna suhte kiire kasv. MIMD arvutite ajalugu on küllaltki pikk ja sisutihe, paljud akadeemilised ja kommertsgrupid on ehitanud multiprotsessorsüsteeme oma aja parimatest mikroprotsessoritest, aga need on alati vajanud spetsiaalseid ühendusmeetodeid. Akadeemilise kogukonna jaoks oli see huvitav uurimisala, aga tulemusena saadud masinad olid enamasti ainukesed omasugused ja ainukeseks ka jäid. Selliste masinate elutsükkel oli tugevas korrelatsioonis neid ehitanud õpetlaste elutsükliga. Riistvaratootjad pakkusid oma spetsiifilisi lahendusi, mis nõudsid programmeerijaid omaks võtma tootjaspetsiifilisi programmeerimismudeleid. See viis tihti tarkvaraarenduse seisukohalt ummikteedele. PC'dele sobivate kõrge jõudlusega võrkude hinna langemine ja Linuxi toetus neile võimaldab nüüd ehitada mõistlikke süsteeme täielikult *COTS* tehnoloogial ning see on teinud praktiliseks ka tavapärased üldised programmeerimismudelid ja arhitektuurid. Linux'i arenemine, GNU tarkvara ja PVM ning MPI standardiseerumine annab programmeerijatele garantii, et nende loodud programmid töötavad ka tulevastel Beowulf klastritel. Baseerumine üldkättesaadavatel tehnoloogiatel murrab sõltuvuse konkreetsetest riistvaratootjatest.

Tõsistele paralleelprogrammeerimise spetsialistidele on Beowulf andnud võimaluse luua väheste kulutustega endale isiklik arvutusplatvorm – neil ei ole enam vajadust jagada superarvuti ressursse suure hulga teiste soovijatega (Beowulf siiski alati ei asenda tõeliselt võimsaid superarvuteid). Beowulf' i klastreid on ehitanud ja kasutanud ka programmeerijad, kes varem pole paralleeltöötlusega üldse kokku puutunud. Beowulf' i klastrid pakuvad näiteks ülikoolidele, kelle ressursid on tavaliselt piiratud, suurepärase platvormi paralleelarvutuste õpetamiseks ning teadusarvutuste läbiviimiseks. Klastri käivituskulud ülikoolis on minimaalsed, kuna asjast huvitatud tudengid ja õppejõud ühendavad arvutid ise kokku ja kirjutavad sellele paralleelprogramme – see on osa õppetööst.

Analoogselt Linux' i kogukonnale on Beowulf' i kogukond nõrgalt organiseeritud ühendus uurijatest ja arendajatest. Igal organisatsioonil on oma eesmärgid ja konkreetsed põhjused, miks arendada teatud osa Beowulf süsteemist. Selle tulemusena varieeruvad Beowulf tüüpi klastrid mõnemasinalistest kobaratest kuni sadu arvuteid sisaldavate süsteemideni.

Enamus inimesi Beowulf' i kogukonnas on iseseisvad oma-lõbuks-arendajad. Kuna igaüks ajab oma asja, ei ole tsentraalsel kontrollil mingit mõtet. Beowulf' i kogukonda hoiab koos liikmete soov jagada oma ideid ning arutada enda edu ning ebaõnnestumisi. Suhtlusmehhanismideks on meililistid, isiklikud võrguleheküljed ja aeg-ajalt toimuvad kohtumised ning *workshop*' id.

Beowulf projekti tuleviku määravad selle arendajate tööpanused projektile ning *COTS* komponentide tulevik. Mikroprotsessorite tehnoloogia areneb ning järjest suurema kiirusega võrgud on laialt kättesaadavad, järjest rohkem rakenduste loojaid hakkavad kasutama paralleelplatvorme ning aegamööda täiustudes täidab Beowulf oma niši.

Kokkuvõte

Paralleelsüsteemide kasutamine näitab maailmas tõusutendetsi. Kõrget jõudlust pakkuvaid süsteeme ehitada on järjest odavam ja lihtsam. Eelkõige on vaja entusiasmi ja huvi asja vastu. Võrku ühendatud arvutitest (nt. arvutiklassis) õppeotstarbelise klastri moodustamisega saaks hakkama iga tavaline tudeng (kasutades näiteks *Bootable Cluster CD*' d).

Arvutikobarate ehitamine on valdkond, mis ühendab endas kõikvõimalikke arvutiteaduse harusid nagu arvutite ja võrkude arhitektuur ning tehnoloogiad, programmeerimiskeeled, interaktsioonide ja kommunikatsiooni probleemid, süsteemide koostamine ning nende efektiivsuse tõstmine jpt. Sellest tulenevalt on arvutiinsenerile igati kasulik klastritega tegeleda (eriti kui tal on vajadus suure arvutusvõimsuse järele). See annab talle ülevaate ka väljaspool oma kitsast spetsialiseerumisvaldkonda paiknevatest ideedest ja probleemistikest.

Kasutatud kirjandus

Cluster Computing. Richard S. Morrison.
(http://www.netspace.net.au/~morri/cct/cct.zip)

Linux HPC Cluster Installation. Luis Ferreira, Gregory Kettmann, Andreas Thomasch, Eileen Silcocks, Jacob Chen, Jean-Claude Daunois, Jens Ihamo, Makoto Harada, Steve Hill, Walter Bernocchi, Egan Ford.
(http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg246041.pdf)

Beowulf HOWTO : System Design.
(http://www.tldp.org/HOWTO/Beowulf-HOWTO-4.html)

Linux Parallel Processing HOWTO.
(http://www.tldp.org/HOWTO/Parallel-Processing-HOWTO.html)

Introduction to Clusters: Build Yourself a PC cluster NOW!
Drs. Christian Halloy & Kwai Wong.
(http://www.jics.utk.edu/SC2001/pcc2001_Oct01_01.pdf)

Beowulf Tutorial: Building a Beowulf System. Jan Lindheim.
(http://www.cacr.caltech.edu/beowulf/tutorial/building.html)

Introduction to the CESDIS Beowulf Project. Phil Merkey.
(http://www.beowulf.org/intro.html)

Getting Started with Debian Beowulf. Adam Powell.
(http://lyre.mit.edu/~powell/debian-howto/beowulf-start.html)

Cornell Students Build Computer Cluster with Castoffs. Brian McDonough.
(http://www.newsfactor.com/perl/printer/17910/)

Dumpster, dumpster spare that junk – it might be a supercomputer. Bill Steele. *(http://www.news.cornell.edu/Chronicle/02/5.16.02/cluster_computer.html)*

High-Performance Interconnects in Cluster Environments.
(http://ftp.us.dell.com/app/4q01-Cti.pdf)

HPC Cluster Interconnects and Message-Passing Systems: From Proprietary to Commodity. Tau Leng, Ph.D.; Rizwan Ali; Christopher Stanton; Jenwei Hsieh, Ph.D. *(http://ftp.us.dell.com/app/4q01-Len.pdf)*

How to Build a Beowulf Linux Cluster.
(http://www.mcsr.olemiss.edu/bookshelf/articles/how_to_build_a_cluster.html)

Linux Cluster Architecture. Alex Vrenios.
(http://www.ncsa.uiuc.edu/~jgreen/LCApres.pdf)

Networks Of Workstations. Daniel Etiemble.
(http://www.eecg.toronto.edu/~de/PA-08.pdf)

The Do-It-Yourself Supercomputer.
William W. Hargrove, Forrest M. Hoffman, Thomas Sterling.
(http://www.sciam.com/print_version.cfm?articleID=000E238B-33EC-1C6F-84A9809EC588EF21)

Virtual Interface (VI) Architecture - The New Open Standard for Distributed Messaging Within a Cluster.
(ftp://ftp.compaq.com/pub/supportinformation/papers/ecg0980998.pdf)

Bootable Cluster CD. *(http://www.cs.uni.edu/~gray/bccd/)*

Inglise - Eesti tehnikasõnaraamat. Euroülikool, 2000.

Arvutikasutaja sõnastik. *(http://ee.www.ee/AKS/AKS.cgi)*

Eesti Keeletehnoloogia sihtasutuse Inglise - Eesti sõnastik. *(http://kiisu.eki.ee)*

E-mail: taivo@vkg.werro.ee