HANDELSHÖGSKOLAN
Vid Göteborgs Universitet
Institutionen för Informatik

2001-09-21

Processmigreringssystem

Ett effektivt sätt att utnyttja ett beräkningskluster?

Processmigrationsystem

An efficient way to use a computational cluster?

D-uppsats VT 2001
Examinator: Alan B Carlson
Handledare: Birgitta Ahlbom
Författare: Andreas Boklund
Fredrik Larsson

Abstract

The construction and use of clusters to replace “supercomputers” are becoming more and more common. The cluster systems that are constructed are often based upon a message passing system or a batch system that statically place the processes on different computers. We believe that the clusters of the future will be based upon some kind of process migrating system. Clusters will then work and look like a single system and behave in the same way as a traditional “supercomputer”.

The purpose of this thesis is to compare two process migrating systems for the Linux platform and investigate what kind of applications they are best suited for. Another purpose is to analyse what strengths/weaknesses they have compared to each other. The application types that we used were processor, memory, hard drive and network intensive applications. Four applications with different characteristics were installed on two clusters based on different process migrating systems. The applications were run on the clusters and interesting values were collected and interpreted.

The process migrating systems of today are not yet fully developed and some functionality is missing. We believe that the process migrating systems have the future ahead of them. One of the process migration systems that we used is usable today, and the other one have been improved since we started to work on this thesis.

Sammanfattning

Det börjar idag bli allt vanligare med konstruktion och användande av klustersystem som kompletterar eller ersätter ”superdatorer”. De klustersystem som konstrueras är oftast baserade på ett meddelandesystem eller batchsystem, som statiskt placerar processer på olika datorer. Vi tror dock att framtidens klustersystem kommer att vara baserade kring någon sort av processmigreringssystem. Klustersystemet kommer då att fungera och se ut som en enda dator och uppföra sig på samma sätt som en traditionell ”superdator”.

Syftet med uppsatsen var att jämföra två processmigreringsystem till Linux plattformen, utreda vilka applikationstyper som de passar för, samt vad de har för styrkor/svagheter jämfört med varandra. Applikationstyperna som definierades var processor, minnes, sekundärlagrings och nätverksintensiva applikationstyper. Fyra applikationer med olika karaktär införskaffades, och två klustersystem med olika processmigreringssystem sattes upp. Applikationerna kördes sedan på klustersystemen och mätvärden samlades in och tolkades.

De processmigreringssystem som finns idag är ännu inte fullt utvecklade och viss funktionalitet saknas. Vi tror dock att processmigreringssystemen har framtiden för sig. Ett av de testade processmigreringssystemen är användbart idag, och det andra har förbättrats sedan vi påbörjade arbetet med denna uppsats.

FÖRORD

Denna uppsats har arbetats fram under våren 2001. Arbetet har varit mycket intressant och lärorikt.

Vi vill börja med att ge ett stort tack till vår handledare Birgitta Ahlbom, som har gett oss många bra synpunkter under arbetets gång. Ett stort tack vill vi även rikta till IT-personalen vid Högskolan i Trollhättan/Uddevalla, då de har bistått med utrustning till den laboratoriemiljö som vi använt vid våra mätningar. Laboratorium för Interaktionsteknologi har även bidragit till denna uppsats genom att ställa två arbetsplatser till förfogande.

Vi vill även passa på att tacka Robi Bandyopadhyay samt Mikael Ericsson som bedriver forskning kring verkstadsindustriprocesser vid HTU.

Denna uppsats hade inte kunnat genomföras utan hjälp från Fluent Inc. och MSC Software med programvara, parallella licenser samt många råd och uppmuntran.

Innan vi undertecknar detta förord vill vi tacka alla som har varit intresserade av vårt arbete, och gett oss många nyttiga och onyttiga kommentarer samt moraliskt stöd.

Handelshögskolan vid Göteborgs universitet september 2001

Andreas Boklund

Fredrik Larsson

Innehållsförteckning

1 INLEDNING

1.1 Bakgrund

Enligt Moores lag fördubblas datorprestandan var tolvte till artonde månad. Trots den snabba utvecklingstakten finns det fortfarande tillämpningar som kräver en högre prestanda än vad som idag finns tillgängligt. Anledningen till detta är kravet på högre precision. Både Cray och IBM har sedan tidigt sjuttiotal kopplat ihop flera processorer i sina superdatorer och låtit dem samarbeta. En superdator är en extremt kraftfull dator med den högsta processorkraften och I/O kapaciteten som är tillgänglig vid den aktuella tidpunkten. Eftersom det är en flytande gräns kan dagens superdator vara morgondagens standarddator (Spector, 2000). Superdatorer används till speciellt krävande tillämpningar, exempelvis simuleringar, väderprognostisering eller filmproduktion.

Superdatorer kostar oftast mycket pengar. En standarddator av idag är mycket kraftfullare än gårdagens superdator. Den snabba utvecklingen har lett till att en superdator av idag är gammal redan när den levereras och en nyare modell kan ha tagit dess plats. Dessutom är uppgraderingsmöjligheterna för en superdator ganska begränsade (Spector, 2000). Med anledning av detta började man se sig om efter nya lösningar för att skapa kraftfulla och billiga datorer.

Det var dock inte förrän 1994 som den första parallelldatorn konstruerad av vanliga standardkomponenter såg dagens ljus, det så kallade Beowulf-klustret vid NASAs Goddard Space Flight Center (beowulf.gsfc.nasa.gov). Det nya var att man skapade en parallelldator genom att koppla samman flera standard PC, utan att använda sig av specialtillverkade komponenter. Beowulf-klustret bestod av 16 stycken Intel 486 baserade datorer sammankopplade med ett Ethernet nätverk. Beowulf-projektet bevisade att det var möjligt att med en liten budget skaffa sig nära nog superdatorprestanda till ett överkomligt pris. (beowulf.gsfc.nasa.gov)

Det var i huvudsak två händelser som möjliggjorde denna utveckling. Konkurrensen på persondatormarknaden hade drivit ner priserna och upp prestandan på datorkomponenter, samt att plattformsoberoende gratisprogramvara med fri källkod hade börjat spridas, såsom GNU från Free Software Foundation och Linux (beowulf.gsfc.nasa.gov). Free Software Foundation är en organisation som arbetar för fri mjukvara. Det som kännetecknar fri mjukvara är att den är gratis och att det är fritt att ändra i källkoden. Detta hanteras enligt de regler som ställs upp av bland annat GNU licensen, vilken innehåller restriktioner på återdistribution av den modifierade källkoden (www.gnu.org). Anledningen till att programvaran anses vara en viktig del är dess plattformsoberoende, det vill säga man är inte tvungen att använda en viss sorts hårdvara för att systemen skall se likadana ut och fungera på samma sätt. Programmerarna behöver då inte ta någon större hänsyn till vilken plattform som de utvecklar sitt program för.

Ytterligare ett projekt som syftar till att skapa ersättare till superdatorer är MOSIX-projektet som drivs av forskare vid The Hebrew University of Jerusalem. MOSIX-projektet var från början en studie av distribuerade operativsystem på högpresterande arbetsstationer. Numera är MOSIX-projektet liksom Beowulf-projektet baserat på sammankopplade standard PC.

Det är enklare att uppgradera ett föråldrat klustersystem än att uppgradera en superdator eftersom man kan lägga till nya noder eller byta ut noder till en ny datorgeneration mer eller mindre under drift (Spector, 2000). Om man bygger ett klustersystem kan man även använda sig av befintliga arbetsstationer, vilket gör att man i vissa lägen kan skaffa sig superdatorlik prestanda utan att göra några större investeringar i hårdvara.

1.2 Problemanalys

Många användare av prestandakrävande mjukvaror använder sig idag av arbetsstationer som trots ett högt pris inte är speciellt kraftfulla. Bland dessa personer/organisationer finns ett behov av att utöka sin datorprestanda men inte resurser att investera i en ”superdator”. Alternativet är att införskaffa ett kluster av standardkomponenter, vilket inte är enkelt eftersom det är en ny teknik samt att mjukvaran ofta måste anpassas och optimeras för den aktuella implementationen. Det är dessutom viktigt att anpassa bland annat processmigreringssystem efter programvara/hårdvara och problem.

För att det skall vara intressant att använda ett klustersystem måste beräkningen som skall utföras vara möjlig att dela upp i ett flertal delproblem. Hårdvarukonstruktionen av klustersystemet bör sedan bland annat baseras på problemets uppdelningsbarhet. Vid ett problem som kan delas upp i åtta lika beräkningskrävande delproblem, finns det ingen anledning att ha fler än åtta noder, däremot bör man skaffa sig de kraftfullaste noderna som ryms i budget. Är problemet delbart i några hundratusen små delproblem kan det löna sig att koppla in så många datorer som möjligt oavsett deras individuella kapacitet. Det som då förmodligen kommer att sätta stopp är nätverkets kapacitet.

1.3 Syfte

Det huvudsakliga syftet med denna uppsats är att jämföra två olika processmigreringssystem till Linux plattformen. Med utgångspunkt i problemanalysen har vi definierat en huvudfråga samt en delfråga vilka vi vill svara på i denna uppsats.

1.3.1 Frågeställning

Vilka applikationstyper lämpar sig de två utvalda processmigreringssystemen för?

De applikationstyperna som vi har valt kan delas in i, processor, minnes, sekundärlagrings och nätverksintensiva applikationer. En delfrågeställning är att undersöka vilka styrkor/svagheter processmigreringssystemen har jämfört med varandra.

1.4 Avgränsning

Läsaren förutsätts ha grundläggande kunskaper om Unix/Linux operativsystem, datorns uppbyggnad, datornätverk samt enkel programmering.

Vi tar endast hänsyn till datorer som är baserade på Intel:s x86 arkitektur. Vi kommer endast att använda oss av ett dedicerat homogent klustersystem (förklaras i 3.2 Klusterarkitekturer).

De slutsatser som vi drar kommer endast att vara relevanta för den hårdvara som har använts i kombination med de versioner av processmigreringssystem och programvaror som använts. Resultaten är även avhängiga på de modeller som har använts och gäller endast för dessa modeller. En modell för termisk sprutning har inte samma karaktär som en för kylning av ett borr vid borrning, även om båda kan beräknas i samma applikation. De modeller som vi har använt är typiska modeller för forskarna som vi har erhållit dem från. Detta innebär att dessa forskare kan applicera resultaten på sina resterande modeller, men resultaten är inte relevanta för andra typer av modeller.

1.5 Disposition

Det första kapitlet efter inledningen är ett metodkapitel. I metodkapitlet (kapitel 2) redogör vi för hur vi har inhämtat informationen som ligger till grund för uppsatsen, samt vilka forskningsmetoder som vi har använt oss av.

Därefter sker en genomgång av de generella hårdvaruarkitekturer som det är nödvändigt att ha kännedom om (kapitel 3). Detta för att få en djupare förståelse för teorierna kring klustersystem i allmänhet och processmigreringssystem i synnerhet. I mjukvaruarkitekturs kapitlet som följer sedan (kapitel 4) hanterar vi först de två processmigreringssystemen, dess funktioner, och sedan de applikationer som vi har använt oss av i laborationerna.

I kapitlet Laboratoriemiljö (kapitel 5) beskrivs hårdvaran som vi har använt samt hur installationen av klustren gick till. I kapitlet därefter (kapitel 6) redogörs de mätresultat som erhölls under laborationerna.

Resultatanalysen (kapitel 7) inleds med en diskussion angående installation och användbarhet. Därefter utvärderas funktionaliteten hos processmigreringssystemen. I det sista kapitlet (kapitel 8) redovisas de slutsatser som vi har kommit fram till, samt förslag till ytterligare forskning inom området.

2 METOD

I detta avsnitt beskrivs de metoder som vi använt. Litteraturstudie av artiklar, rapporter samt observationer och laboratorieförsök, är instrument som har hjälpt oss att förstå och tolka materialet som kommit ut av vår undersökning. Vi har delat upp vårt arbete i tre steg:

- Litteraturstudie
- Laboratorieförsök
- Resultatanalys

2.1 Forskningsmetod/angreppssätt

Vi har använt oss av en kombination av kvalitativ och kvantitativ metod där vi först kartlade skillnaderna mellan de två systemen. Därefter analyserades resultaten från testkörningarna av de utvalda applikationerna. Kvantitativ och kvalitativ forskningsmetod framställs ofta som om de vore oförenliga, vilket oftast inte är fallet vid praktiskt forskningsarbete. En stor del av den forskning som bedrivs är svår att klassificera och använder sig ofta av båda metoderna. (Patel & Davidson, 1994)

Det är vanligt att kvantitativt inriktad forskning i analysen använder sig av beskrivande termer, det vill säga mjuka data. Det är likväl vanligt att kvalitativ forskning ofta har inslag av statistiska analyser, det vill säga hårda data. (Widerheim-Paul & Eriksson, 1991)

Vi anser att en explorativ undersökningsmetod stämmer bäst överens med vår undersökning. Det främsta syftet med en explorativ undersökning är att inhämta så mycket kunskap som möjligt om ett visst ämne, för att belysa det allsidigt. Den explorativa undersökningsmetoden är utforskande och ligger ofta till grund för vidare studier inom ämnet. Idérikedom och kreativitet är viktiga inslag i en explorativ undersökning. Vid explorativa undersökningar använder man sig ofta av olika tekniker för att samla in information. (Patel & Davidson, 1994)

2.2 Litteraturstudie

Informationen som vi använt oss av i detta arbete kommer mestadels från forskningsrapporter, och olika källor på Internet. Detta beror på att det än så länge inte finns så mycket litteratur inom ämnet. De källor på Internet som använts är i de flesta fallen någon organisation (grupp av forskare eller företag) som genomfört ett klusterprojekt och sedan delat med sig av resultaten till intresserade.

Som en ytterligare informationskälla har vi använt de mailinglistor (bland annat beowulf@beowulf.gsfc.nasa.gov) som vi varit med i under en längre tid, då det där finns mycket information samt stor kompetens.

2.3 Konstruktion av ett kluster

Vi konstruerade två fungerande kluster genom att koppla samman ett antal datorer. En dator fungerade som huvudnod, det vill säga den fördelade arbetet mellan noderna samt sammanställde resultatet. Det är endast denna dator som användaren av klustret arbetar med. Övriga datorer registrerar sig vid uppstart hos huvudnoden och därefter kan de börja utföra beräkningar. Det finns egentligen två anledningar till denna konstruktion. Den första är att vi behövde ett klustersystem att utföra våra laborationer på. Den andra var att under konstruktionsarbetet tillskansa oss djupare kunskap om tekniken och logiken bakom hur ett kluster fungerar, än vad som kan erhållas genom en litteraturstudie.

2.4 Mätningar

Vi har valt att genomföra undersökningen med hjälp av fyra applikationer med olika parallell struktur, och studera hur de båda processmigreringssystemen hanterar dem. De applikationer som vi använde oss av var: Fluent, kompilering av Linux kärna, MSC.Marc samt PovRay. Anledningen till att vi valde att använda oss av Fluent och MSC.Marc är att de används av forskare i vår närhet, samt att vi har goda kontakter med företagen bakom respektive programvara, och därigenom fått tillgång till tidsbegränsade användarlicenser utan kostnad. PovRay användes för att den är känd för att vara ett exempel på en extremt parallell applikation. Kompileringar av Linux kärnan är en tidskrävande process som ibland används i utvärderingssyften.

Den variabel som är viktigast för användarna är exekveringstid, detta på grund av att det som eftersträvas vid användandet av superdatorer och kluster är att förkorta ledtiderna för vissa uppgifter. Därför vill man ofta finna de flaskhalsar som finns i ett klustersystem.

Tre identiska mätningar utfördes och ett medelvärde av dessa mätningar användes. Värdena som samlades in var tidsåtgång, processor-, minne- samt nätverksutnyttjande. Detta gjordes var tionde sekund på huvudnoden samt på en av noderna, dock alltid på samma nod.

Mätningarna genomfördes för att se hur de två processmigreringssystemen hanterar de flaskhalsar som finns samt om de introducerar några nya. Anledningen till att applikationerna kördes flera gånger över olika antal noder var bland annat för att se om en hastighetsökning erhålls om man använder sig av ytterligare noder, och i så fall hur stor denna är. Detta då det är möjligt att ett av processmigreringssystemen eller applikationen i fråga inte kan hantera mer än ett visst antal noder.

Alla applikationer genererar loggfiler som visar bland annat tidsåtgång i sekunder. Under mätningarna mättes nätbelastningen med hjälp av de loggningsfunktioner som fanns i den switch vi använde. Mätresultaten kommer att diskuteras i kapitlet laboratorieförsök.

3 HÅRDVARUARKITEKTUR

3.1 Datorarkitektur

De flesta av dagens datorer har minst en processor. Det blir dock vanligare att arbetsstationer, servrar och beräkningsdatorer är utrustade med flera processorer vardera. Det finns ett antal olika sätt att låta flera processorer samarbeta. De fyra vanligaste är Massively Parallel Processors (MPP), Symmetric Multi Processing (SMP), dedicerat kluster samt ickededicerat kluster. Det finns dock inga hinder för att använda sig av SMP eller MPP datorer i en klusterlösning. (Buyya, 1999a)

MPP är en stor parallelldator som oftast består av flera hundra beräkningsenheter (noder) vilka oftast endast innehåller en processor och internminne. Det kan dock i vissa fall finnas specialnoder med exempelvis hårddiskar eller backupenheter. Noderna är sammankopplade via ett höghastighetsinterface. Den utmärkande skillnaden mellan en MPP och ett kluster är att MPP noderna delar på variabler och kan utnyttja varandras internminne. (Buyya, 1999a)

SMP innebär att man har två eller flera processorer av samma typ på samma moderkort (eller processorkort), och att de använder sig av samma chipset (kontrollkretsar) och internminne. SMP är numera vanligt förekommande i kraftfulla servrar och arbetsstationer. (www.linuxdoc.org)

Ett dedicerat kluster består av två eller flera datorer som är sammankopplade i ett nätverk. En av datorerna fungerar som huvudnod. Huvudnoden är den dator som kör själva applikationen och fördelar arbetet över de noder som finns i klustret. I ett dedicerat kluster är det vanligtvis endast huvudnoden som har tillgång till ett externt nätverk. Noderna i ett dedicerat kluster blir oftast billigare eftersom de inte kräver grafikkort, skärm, tangentbord och mus, då all eventuell konfigurering kan göras från huvudnoden via nätverket.

Ett ickededicerat kluster består även det av ett flertal datorer sammankopplade i ett nätverk och kan ha samma topologi som ett dedicerat kluster. Skillnaden mot det dedicerade klustret är att noderna här även fungerar som vanliga arbetsstationer, och endast överskottskapacitet används till de beräkningar som huvudnoden delar ut till dem. Denna lösning innebär i de flesta fall att klusterarbetet utförs utan att användaren av noden märker någonting. Fördelen med denna lösning är att man kan använda sig av befintlig hårdvara och därmed inte behöver göra stora nyinvesteringar i utrustning.

Enligt Amdahls lag (från 1967) består ett program av två olika sorters av beräkningar, sådana som måste utföras seriellt, och sådana som kan parallelliseras för att köras över flera processorer (Wilson, 1995).

Amdahls lag

> Antag att tiden som det tar att beräkna den seriella delen är Y, och att tiden som det tar att beräkna den parallella delen är X. Antag sedan att den parallella delen är linjärt skalbar över antalet processorer. Detta innebär att oberoende av hur många processorer som används kommer problemet aldrig att kunna lösas fortare än Y tidsenheter. (Wilson, 1995)

3.2 Klusterarkitekturer

Det som eftersträvas när man konstruerar ett kluster är att det skall vara så likt en SMP eller en MPP arkitektur som möjligt. Detta kan åstadkommas både i hårdvaru- och mjukvarukonfigurationen. (Buyya, 1999a)

Ett kluster kan hårdvarumässigt konstrueras på ett flertal olika sätt, dessa delas dock oftast in i två huvudgrupper, homogen eller heterogen arkitektur. Ett homogent kluster består av noder som alla har exakt samma hårdvara. I ett kluster med en heterogen arkitektur kan alla noder ha olika hårdvaru- och mjukvaruspecifikationer. Fördelen med att använda sig av samma hårdvaru- och mjukvarukonfiguration, det vill säga ett homogent kluster, är att konstruktionen av klustret förenklas eftersom alla maskiner är likadana. I ett heterogent kluster kan det finnas behov av ingående kunskaper om exempelvis drivrutiner för flera plattformar och ordentliga kunskaper om många operativsystem.

Beroende på problemstrukturen kan det finnas ett behov av att välja antingen en homogen eller en heterogen arkitektur (Brown, 2001). En viktig faktor är lastbalansering; det vill säga hur beräkningarna skall fördelas över noderna beroende på de olika nodernas belastning och kapacitet. En del problem går att dela upp i ett antal lika beräkningskrävande seriella delar, medan andra kan bestå av exempelvis maximalt sex seriella delar eller av en stor seriell del samt flera mindre. Detta är faktorer som bör utredas innan man bestämmer sig för att konstruera ett kluster för att köra en specifik applikation, det finns stora möjligheter att ett kluster som är bra på att lösa ett problem inte alls är anpassat för att lösa ett annat. (Boklund & Larsson, 2001)

> *Kluster kan medvetet göras heterogena för att man skall kunna utnyttja maximal prestanda. Ingenting hindrar dig från att koppla in en superdator som en nod.*
>
> *-- Buyya, 1999a*

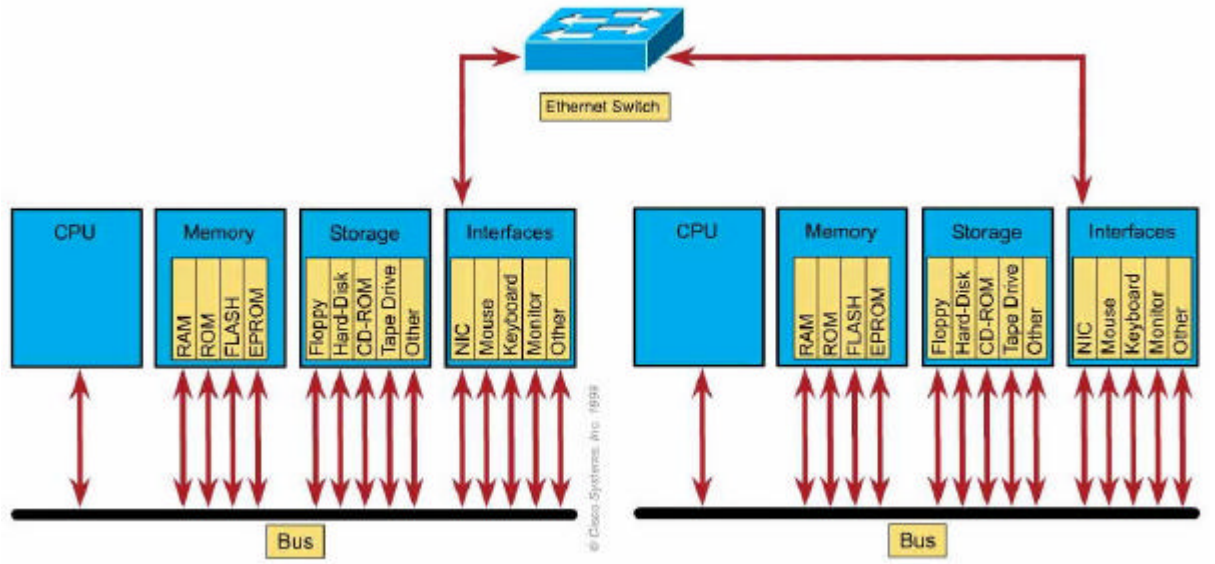
En fördel med en heterogen klusterarkitektur är att det är enklare att lägga till och dra ifrån noder med tanke på att ingen hänsyn behöver tas till hårdvaran. En nackdel är att det blir svårare att administrera och underhålla ett heterogent kluster. (Buyya, 1999a)

Ett kluster kan vara antingen dedicerat eller ickededicerat. Ett dedicerat kluster används endast till att utföra den uppgift som för tillfället har ålagts den. I ett ickededicerat kluster är det möjligt att en eller flera av noderna används som arbetsstationer, eller fungerar som servrar åt någon annan verksamhet. Homogena kluster är uteslutande även dedicerade. Detta beror på att om noderna används till annat arbete står inte hela klustrets prestanda till huvudnodens förfogande. (Boklund & Larsson, 2000)

3.3 Klustrets moduler

Ett av de största problemen med att konstruera ett klustersystem är att skapa ett så effektivt klustersystem som möjligt utan att över eller underdimensionera någon del. En överdimensionering leder till ökade kostnader medan en underdimensionering leder till en prestandaförlust. I båda fallen blir resultatet ett slöseri av resurser. Vidare så är vissa applikationer mer resurskrävande än andra. Det är dock oftast så att en applikation har större krav på en viss del i klustersystemet än på en annan del.

Enligt Hawick, Grove, Coddington, och Buntine (2000) vid Department of Computer Science, University of Adelaide kan man dela in ett kluster i fyra viktiga delar. De delar som de nämner är: processorhastighet, RAM minne, hårddiskar samt nätverk (Hawick et al., 2000). Buyya anser dock att även datorernas systembuss och processorcache bör tas hänsyn till (Buyya, 1999a). Med bakgrund av detta samt Robert G. Brown, vid Duke University Physics Departments teorier kring klustersystem och dess flaskhalsar (Brown, 2001) har vi valt att dela in klustersystemet i fem moduler. Nedan är en schematisk skiss av modulerna samt en kortare beskrivning av varje modul.



Figur: 3.1 Schematisk skiss över klustrets moduler (Modifierad: cisco.netacad.net)

3.3.1 Processortyp

Det finns idag många processormodeller exempelvis; Sun SPARC, Compac Alpha, IBM PowerPC, SGI Mips, Intel x86 samt x86 kloner. Under de senaste åren har processorernas klockfrekvens blivit allt högre. En processors klockfrekvens kopplas ofta direkt samman med hur snabb processorn är och används i vissa fall även som ett mått på hela datorns prestanda. Detta är inte gångbart. Det fungerar ofta att använda klockfrekvensen som ett riktmärke men det är inte alltid rättvisande. De aspekter hos en processor som bör tas hänsyn till är dess beräkningsprestanda vad det gäller heltalsoperationer (MIPS), flyttalsoperationer (FLOPS) samt storlek och hastighet på processorns cacheminnen. Applikationer har olika krav på processorns olika delsystem (Brown, 2001). I resultatdelen kommer vi endast bedömma om applikationerna är processorintensiva eller ej.

3.3.2 Arbetsminne

Det är inte bara storleken på klustersystemets RAM minne som är intressant när man konstruerar ett klustersystem, utan även vilken typ av minne som man använder sig av. Olika sorters minne arbetar på olika sätt, de vanligaste minnestyperna idag är SDRAM och RDRAM. RDRAM har en längre söktid än SDRAM men även större bandbredd. RDRAM är för tillfället också betydligt dyrare än SDRAM (www.tomshardware.com).

Den mängd minne som klustersystemet behöver beror på applikationernas behov. Applikationer som är parallelliserade borde distribueras på ett sådant sätt att minnesbehovet fördelas mellan huvudnoden och noderna. Det finns därför inget behov av att ha nog med arbetsminne i varje nod för att klara av att hålla hela applikationen i minnet (Buyya, 1999a). Det är däremot möjligt att man har mindre arbetsminne i noderna än vad applikationerna kräver, då vissa applikationer kräver hundratals megabyte av RAM minne eller mer på varje nod. Om datorernas arbetsminne fylls är det viktigt att det finns möjligheter för datorerna att använda sig av virtuellt minne, så kallat swapminne. Detta innebär att operativsystemet skriver ner en del av arbetsminnet på hårddisken för att sedan kunna läsa det därifrån när det behövs. När en dator använder sig av virtuellt minne försämras prestandan avsevärt och applikationens hastighet sänks till minst en tiondel i dagens system (Brown, 2001). Man bör i största möjliga mån se till att datorerna har nog med arbetsminne för att hantera de applikationer som man tänker använda sig av. Problemet är att arbetsminne är dyrt och i vissa fall kan minneskostnaden bli den största kostnaden för klustersystemet. Används virtuellt minne så bör man se till att avdela ordentligt med plats på hårddisken samt använda sig av en snabb hårddisk. Om en applikation har ett stort behov av arbetsminne har det även ett stort behov av snabb åtkomst till minnet, därför kommer vi i resultatdelen att bedöma arbetsminnet utefter storleken på applikationens behov.

3.3.3 Sekundärlagringsenhet

Med sekundär lagringsenhet menas oftast en dators hårddisk. Det måste dock inte vara en hårddisk utan kan lika gärna vara disketter, zipskivor, bandstationer eller NFS monterade volymer (NFS, Network File System). Det finns egentligen två ytterligheter när det gäller klustersystem och sekundäralagringsenheter. Den ena ytterligheten är att alla noder har var sitt snabbt SCSI-RAID system och den andra är att noderna servas av huvudnoden eller en separat filserver. Det är dock vanligast med en gyllene medelväg där alla noder har var sin billig IDE hårddisk, som används till operativsystem och virtuellt minne. Dessutom NFS-monteras applikations och arbetsfiler från huvudnoden eller en filserver, som ofta är utrustad med snabba och stora SCSI hårddiskar. (Hunt, 1998)

Den stora fördelen med att inte ha några hårddiskar i noderna är att även billiga IDE hårddiskar kostar pengar. Att montera snabba SCSI hårddiskar i noderna, eller i varje fall i huvudnoden, är intressant om applikationerna läser eller skriver mycket till hårddisken eller om de har ett stort behov av arbetsminne och därför behöver använda sig av virtuellt minne. Det kan då vara ett alternativ eftersom SCSI hårddiskar är snabbare och mindre processorkrävande än IDE hårddiskar samtidigt som de är markant billigare än arbetsminne. I resultatdelen kommer vi endast att ta hänsyn till om applikationerna är diskintensiva eller ej, det vill säga om de läser/skriver mycket till sina lokala hårddiskar.

3.3.4 Nätverk

Ofta när man talar om klustersystem talar man om huvudnod, noder och nätverk. Nätverket är en viktig del av ett klustersystem. Ju snabbare nätverk, desto snabbare utför applikationerna sina beräkningar, även om ökningen i vissa fall knappt är märkbar. De flesta klusternätverken idag använder sig av antingen Ethernet, Fast Ethernet eller Gigabit Ethernet, med antingen 10, 100 eller 1000 Mbit/s överföringshastighet. De som kräver mer investerar oftast i ett Myrinet nätverk vilket tillverkas med en hastighet upp emot 2000 Mbit/s (www.myrinet.com).

Beroende på antalet noder och problemets uppbyggnad ställs olika krav på nätverket. I vissa fall kan det vara fullt tillräckligt med ett Ethernet (10 Mbits), medan det i andra fall kan krävas kraftfullare nät än vad som idag finns tillgängligt. Dock finns det idag ingen anledning att vid konstruktionen av ett kluster använda sig av ett vanligt Ethernet eller hubbat nätverk, då en investering i ett Fast Ethernet samt switchar inte innebär någon betydande kostnadsökning (Buyya, 1999a). Det finns två punkter där det är intressant att mäta, dessa är huvudnodens samt en nods trafik. Då huvudnoden nästan alltid har en högre belastning än de enskilda noderna är det inte helt ovanligt att noderna delas i två segment med var sin koppling till huvudnoden. Man kan också ge huvudnoden ett kraftfullare nätverkskort, exempelvis ett Gigabit Eternet kort om noderna är utrustade med Fast Ethernet. Det är även viktigt att nätverkets knytpunkt, vilket oftast är en switch, är anpassad för att kunna klara av trafiken från huvudnoden samt noderna. (Spector, 2000)

Kravet på nätprestanda bör utredas noga innan man konstruerar ett kluster. Formeln som bör användas är:

Antal noder * (delproblem per tidsenhet och nod * nättrafik per delproblem) =
= *nätbelastning per tidsenhet (t.ex. KByte/s)*

Källa: Parallelldatorsystem (Boklund & Larsson, 2000)

Exempel:
Ett kluster har 24 noder, varje nod klarar av att beräkna 100 delproblem/sekund och varje delproblem genererar 1 KByte nättrafik.
Nätbelastningen blir då 24 *(100*1) = 2400 KByte/sekund = 2,4 MByte/s Ovanstående belastning motsvarar ca 24 Mbit/s nättrafik till/från huvudnoden och ca 1 Mbit/s till/från var och en av noderna. Detta innebär att noderna klarar sig med Ethernet (10 Mbit/s) näthastighet medan huvudnoden behöver Fast Ethernet (100 Mbit/s) förutsatt att man använder sig av ett switchat nät. (Boklund & Larsson, 2000)

Det intressanta med nätverket är som sagt hur mycket trafik som applikationen genererar, samt om det ligger ett högre krav på huvudnoden än på noderna. Detta är vad vi kommer att ta hänsyn till i resultatdelen.

3.3.5 Systembuss

Systembussen är den del av datorn som kopplar samman ”alla” de andra enheterna. I en Intel x86 systemarkitektur sker all kommunikation mellan processor, arbetsminne, hårddiskar och nätverkskort över systembussen. Systembussen kan därför utgöra en flaskhals. De idag vanligaste hastigheterna är 100/133Mhz (Intel Pentium III), 2*100Mhz (AMD Athlon) och 4*100Mhz (Intel Pentium 4) (www.tomshardware.com). Som synes är det en stor skillnad i bandbredd men en liten skillnad i latens mellan de olika systemen. Det är dock så att alla enheter förutom processorn och i vissa fall arbetsminnet, vilka är sammankopplade med systembussen, har en lägre hastighet än systembussen. Detta innebär att den endast kan utgöra en flaskhals mot dessa enheter om den använder flera utav dem samtidigt.

Vid användningen av SMP datorer, det vill säga datorer med mer än en processor är det dock intressantare eftersom alla processorer använder sig av samma systembuss, arbetsminne och så vidare. Att använda sig av två processorer i samma dator är oftast något billigare än att köpa två stycken singelprocessordatorer, speciellt med tanke på periferienheter, som till exempel låda, hårddiskar, nätverkskort och så vidare. Om applikationen är rent processorberoende kan man få nästan 100 % hastighetsökning med två processorer jämfört med en. Detta är dock inte alltid fallet. Ibland har applikationerna till exempel ett högt minnesutnyttjande, och i så fall kommer inte båda processorerna att kunna läsa från minnet samtidigt. Det beror både på minnets hastighet och att systembussen inte kan transportera informationen fort nog mellan minne och processorer, vilket även gäller alla andra delar i datorn. Om applikationerna är minnesintensiva blir inte hastighetsökningen 100 % utan troligen runt 40-60 %, och då är det oftast bättre att spendera pengarna på att köpa singelprocessordatorer. (Brown, 2001)

3.3.6 Beroendeförhållanden

Ovanstående moduler har alla beroendeförhållanden sinsemellan. Dessa beroendeförhållanden är olika stora mellan olika moduler. Ett exempel på ett beroendeförhållande är att om man sätter in ytterligare en processor (CPU) i en dator så behöver systembussen vara dubbelt så ”bred” för att den extra processorn skall kunna utnyttjas till fullo (det vill säga CPU:n har ett beroendeförhållande till systembussen). Om man däremot placerar en extra hårddisk i en dator har detta ingen eller liten inverkan på nätverkets belastning. Alla enheter är dock beroende av systembussens kapacitet.

	CPU	Minne	HD	Nät	Buss
CPU		**	*	*	**
Minne					**
HD	*				**
Nät	*				**
Buss					

* Litet beroendeförhållande
** Stort beroendeförhållande

Figur: 3.2 Beroendeförhållande mellan modulerna.

4 MJUKVARUARKITEKTUR

4.1 Processmigreringssystem

Klustersystem baserat på standardkomponenter använder sig av olika tekniker för att distribuera arbetet mellan olika noder. Bland dessa olika metoder återfinns bland annat processmigrering, meddelandesystem och batchsystem. Dessa tre har helt olika användningsområde och kan därför kombineras.

Processmigreringssystem implementeras på operativsystemsnivå. Därför kräver installationen av ett processmigreringssystem att man kompilerar om operativsystemets kärna (kernel). Ett processmigreringssystem kan därför klassas som ett eget operativsystem. Det som särskiljer processmigreringssystem från de andra varianterna av disribueringstekniker är att det som eftersträvas är att skapa en ”Single System Image” (SSI). Det som är kännetecknande för en Single System Image är att en grupp sammankopplade datorer, ett klustersystem, är designat för att framstå som en enhet. Grundtanken med processmigreringssystem är att fristående datorer skall samarbeta som om de vore en enda enhet, det vill säga som en MPP. (Buyya, 1999a)

När en applikation startas skapas en eller flera processer på den dator som applikationen startas på. Processmigreringssystemen flyttar sedan delar av dessa processer, oftast bestående av processdata och processens minnessegment, till andra datorer inom klustersystemet. (Barak, Guday & Wheeler, 1993) Den dator på vilken processen initierades (startades) håller reda på vilken annan dator som processen befinner sig på samt vidarebefordrar systemanrop mellan processer och operativsystemet. Processerna som har blivit distribuerade luras därigenom att tro att de fortfarande befinner sig på den dator där de först initierades. Systemanrop hemmahörande på den initierande datorn kommer därför fortfarande att utföras på denna, exempel på sådana systemanrop är läsning från/skrivning till filer samt datautbyte med andra processer. (Buyya, 1999a) Olika processmigreringssystem hanterar detta på olika sätt.

4.1.1 MOSIX

MOSIX står för **M**ulticomputer **O**perating **S**ystem for UN**IX** och är ett processmigreringssystem utvecklat vid The Hebrew University i Jerusalem, Israel. Arbetet leds av Professor Amnon Barak. Arbetet med den första versionen av MOSIX påbörjades 1981 och utvecklades till fyra Digital PDP/11 datorer. Utvecklingsplattformen har sedan ändrats sju gånger, den senaste av dessa är Linux och Intel x86 plattformen. En gemensam nämnare mellan alla versioner av MOSIX är att de har baserats på en AT&T kompatibel version av UNIX.

MOSIX, som är ett tilläggspaket till Linux kärnan kan laddas ner från MOSIX egen hemsida och är fritt för alla att använda. Projektet finansieras bland annat av Israel försvarsdepartement och USA:s flygvapen.

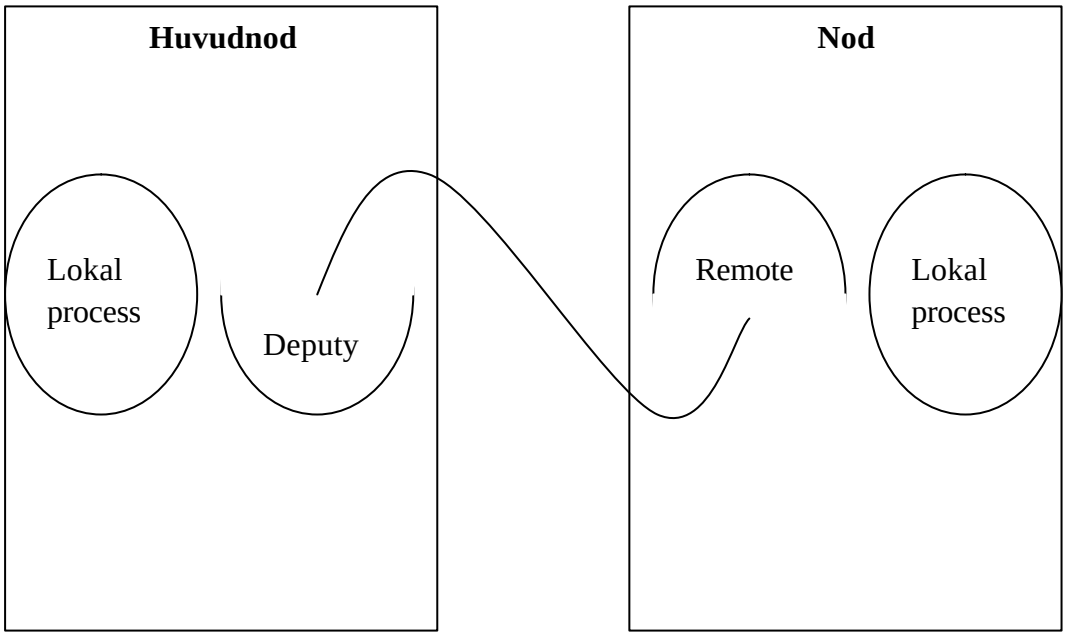
Målet med MOSIX-projektet är att få datorer som endast delar ett nätverk att i så stor utsträckning som möjligt efterlikna en SMP eller MPP arkitektur. De vanligaste klusterarkitekturerna går ut på att man statiskt grupperar ihop processer och hårdvara, vilket ofta är ett tidsödande arbete som cementerar fast vissa strukturer. MOSIX däremot är designat för att läsa av ett klusters tillgängliga resurser och sedan placera processerna där det finns tillräckligt med resurser. Enligt Barak består MOSIX av adaptiva resursdelande algoritmer, som delar processerna mellan noderna beroende på deras individuella konfiguration och belastning. (Barak, La'adan & Shiloh, 1999)

Om man jämför ett kluster med en SMP eller MPP dator så består ofta klustren i större utsträckning av datorer med olika hårdvara. I ett kluster är det inte ens säkert att alla noderna använder sig av samma operativsystem. MOSIX kräver dock att alla datorer använder samma operativsystem. Rekommendationen är även att i så stor utsträckning som möjligt använda sig av samma MOSIX version, då vissa versioner av MOSIX inte fungerar tillsammans.

MOSIX värderar en dators resurser enligt en förutbestämd mall. Alla processorers prestanda jämförs med en Pentium III 450Mhz vilken har riktvärdet 1000. Andra värden som MOSIX tar hänsyn till är exempelvis storleken på datorns RAM minne samt hur stor del av det som är oanvänt. MOSIX skapar dock inga tabeller i vanlig ordning över dessa värden utan använder sig av speciella algoritmer för att gå ut till olika noder och läsa av deras värden. Dessa algoritmer är konstruerade på ett sådant sätt att tiden för avsökning av nodernas värden inte skall öka i takt med antalet noder utan istället vara konstant. (Barak et al., 1993)

När huvudnoden eller en nod, beroende på konfiguration, sedan är överbelastad, ser den efter om någon annan dator i klustret har nog med lediga resurser för att ta över en del av dess arbete. Notera dock att MOSIX arbetar på processnivå vilket innebär att den bara kan flytta hela processer.

Alla processer har var sin Unik Hemma Nod (UHN), detta är den nod på vilken processen skapades. När MOSIX flyttar processen till en annan nod skapar den en så kallad *deputy* process på hemmanoden. Den flyttade processen håller kontakten med sin *deputy* process oavsett vilken nod som den för tillfället befinner sig på. När processen behöver utföra funktioner som är beroende av hemmanoden, till exempel att läsa från eller skriva till en lokal inställningsfil sker detta genom *deputy* processen. Om användaren listar de processer som körs på hemmanoden hämtar operativsystemet data om processen från *deputy* processen, vilken i sin tur har inskaffat informationen från sin utflyttade process. Detta kan vara lite konfunderande om man inte känner till det, och ser att de processer som körs tar mer resurser i anspråk än vad som finns i den dator som man sitter vid.



Figur 4.1: Interaktion mellan MOSIX deputy och remote processer
(Barak et al., 1999)

Användandet av en tvådelad processtruktur (deputy-remote) är i MOSIX fall en nödvändighet. Processerna behöver då inte ta någon hänsyn till att den har flyttats till en annan dator. Det behövs inte heller något lokalt register över alla processer som finns eller var de befinner sig, den informationen samlas dock in från deputy processerna vid de tillfällen som de är åtråvärda. En nackdel med att använda sig av deputyprocesser är att det skapar en del merarbete i form av kommunikation som vanligtvis inte skulle behövas (mellan deputy och remote processerna). Denna extra kommunikation sker över nätverket mellan noderna/huvudnoden, vilket tar både tid och resurser i anspråk.

4.1.2 Scyld

Scyld (version 27bz6) paketet har utvecklats av samma personer som startade Beowulf projektet på NASA 1994, och säljs av företaget med samma namn. Version 27bz6 är den första versionen av Scyld och den enda som fanns tillgänglig i november 2000. Anledningen till att distributionen heter Scyld är att Scyld var Beowulfs far i sagan om Beowulf. Sagan om Beowulf är den äldsta kända boken på engelska. (Beowulf, 2000)

Scyld är en så kallad Linux distribution. Scyld tillhandahåller en förändrad Linux kärna samt verktyg för att använda och administrera ett kluster av datorer. Målet med Scyld är att låta ett Linux kluster se ut som en SMP eller en MPP arkitektur. Scyld har gått ett steg längre än MOSIX genom att ta bort mer eller mindre all funktionalitet från noderna. Detta innebär att Scyld endast lämpar sig för dedicerade kluster, det finns dock inga hinder för en heterogen arkitektur.

Ett Scyld kluster har en huvudnod som tillhandahåller fildelning åt alla noder, eftersom noderna i sig inte har något behov av interna hårddiskar. Noderna i ett Scyld kluster startas från Scylds start CD och all data (linux kärna, bibliotek och verktyg) laddas sedan ner över nätverket. (Scyld Beowulf Reference Manual, 2001)

Processmigrering under Scyld är inte genomskinlig (till skillnad från MOSIX), det vill säga man måste i förväg när man skriver programmet själv implementera en algoritm för processmigrering. Problemet som då uppstår är att alla applikationer måste specialskrivas för att fungera under ett Scyld kluster. Detta går dock att komma runt genom att använda sig av meddelandesystemet BeoMPI, som är en implementation av MPI standarden och använder sig av Scylds processmigreringsimplementation. (www.scyld.com)

För processdelningen använder sig Scyld av ett system som kallas BProc (Beowulf Processes). BProc fungerar på ett sådant sätt att en process som skall köras på klustret startas på huvudnoden. Operativsystemet stoppar sedan processen, kopierar all dess data till en av noderna med hjälp av ett koncept som kallas VMADump (Virtual Memory Area Dump). Processen startas sedan igen och fortsätter med att utföra sitt arbete på noddatorn. Kvar på huvudnoden blir en skalprocess vilken innehåller det som blir kvar av den kopierade processen. En kontrollprocess på huvudnoden håller reda på vilka noder som ingår i klustret, vilka processer som finns samt vilken dator som de körs på. Alla processer som flyttas ut behåller sitt process ID från huvudnoden men erhåller även ett nytt process ID på den nod där de körs. Noderna kör alltså en eller flera av huvudnodens processer. Om dessa processer skapar egna underprocesser kommer även de att köras på noden. (Hendriks, 1999)

4.2 Meddelandesystem

Ett meddelandesystem är ett program som installeras över operativsystemet och har till uppgift att skicka meddelanden/kommandon från en nod till en annan. Meddelandesystem kan användas på två olika sätt, antingen genom att använda sig av programbibliotek för kommunikation eller genom att starta fristående applikationer på de olika noderna. Genom att använda sig av meddelandesystemets programbibliotek när man skriver program, placerar man kommunikationsrutinerna i källkoden. Den senare metoden är självklart den mest effektiva, men kräver dock ingående kunskaper i hur man programmerar för en parallellarkitektur. För att starta fristående applikationer på noderna krävs endast grundläggande kunskaper i UNIX och programmering. (www.epm.ornl.gov)

En viktig del i ett klustersystems framåtkompatibilitet är användningen av ett standardiserat meddelandesystem. Dessa program/bibliotek innehåller funktioner för att dela upp arbetet mellan noderna i ett kluster (Wilson, 1995). Genom att använda sig av ett tredjepartsutvecklat meddelandesystem kan man försäkra sig om att applikationerna kommer att fungera på framtida klustersystem och parallelldatorer oavsett hårdvarukonfiguration.

Det finns idag två stora varianter på meddelandesystem, dessa är MPI-standarden och PVM. MPI standarden skapades och uppdateras av en kommitté som består av hård-, mjukvarutillverkare, universitet samt användare, medan PVM skapades på grund av att det fanns ett praktiskt behov av ett program som kunde förmedla meddelanden mellan noder i ett klustersystem. PVM blev tidigt en de facto standard (Buyya, 1999b).

MPI standarden skapades inte för klustersystem utan för intern kommunikation på datorer som har flera processorer. Det finns därför MPI-implementationer till de flesta superdatorer, exempelvis Cray T3D, Intel Paragon och IBM SP/2. (www-unix.mcs.anl.gov) PVM finns tillgängligt för de flesta Unix-dialekter, samt i versioner för bland annat Windows NT, och går även att använda på SMP och MPP datorer. PVM utvecklas vid Oak Ridge National Laboratory.

4.3 Batchsystem

Ett batchsystem är en utveckling av ett kösystem. Batchsystemen används till att fördela arbete antingen i tiden eller över ett antal datorer. Grunden i alla batchsystem är ett kösystem i vilket användarna kan placera sina uppgifter. Den enklaste varianten av kösystem utför den första uppgiften i kön och när den är färdig utförs nästa uppgift i kön och så vidare. Mer avancerade kösystem tillför ytterligare funktionalitet, detta kan exempelvis vara prioritering av en viss användares uppgift eller att en viss uppgift skall utföras vid en viss tidpunkt.

4.4 Applikationer

4.4.1 Fluent

Fluent (version 5.3) är ett flödessimuleringsprogram som för att utföra de tidskrävande CFD-beräkningarna (Computerized Flow Modeling) använder sig av FEM (Finita ElementMetoden). FEM består av en mängd komplicerade beräkningar, vilka går att dela in i delproblem. (Samuelsson & Wiberg, 1988) Detta innebär att beräkningarna går att utföra på ett klustersystem. Simuleringar kan ersätta många dyra praktiska experiment, som till exempel termisk sprutning av flyg/raket-motorkomponenter. Det tar idag flera dygn att utföra även de minsta simuleringarna på en vanlig arbetsstation, och det är därför av intresse att korta ner tidsåtgången för beräkningsprocessen.

FEM lösningar beräknas med hjälp av en matris. Med hjälp av FEM beräkningarna försöker man uppnå så hög precision som möjligt. För att uppnå precision ”itereras” (upprepas) beräkningarna ett stort antal gånger. Det man försöker räkna ut är förändringar i flöden och flödestryck. FEM kan användas för att utvärdera alternativa lösningar, optimera konstruktioner och kan som tidigare nämnts ersätta dyra praktiska experiment med datorsimuleringar. (Samuelsson & Wiberg, 1988)

Fluent har programmerats med tanke på multiprocessor arkitekturer, anledningen till detta är att de flesta stordatorer/superdatorer är byggda kring någon sorts parallellarkitektur. På grund av att samma meddelandesystem fungerar på superdatorer som kluster, passar Fluent även bra för körning på kluster. Det krävs dock hög beräkningsprestanda för att kunna utföra dessa beräkningar inom en rimlig tid.

Fluent används bland annat av Högskolan i Trollhättan/Uddevalla samt Volvo Aero. Våra mätningar har utförts på en verklig modell som erhållits av Robi Bandyopadhyay som forskar inom simuleringsteknik vid Högskolan i Trollhättan/Uddevalla.

4.4.2 Kompilering

Att kompilera programkod är något som alla som programmerar gör med jämna mellanrum. Att kompilera innebär att man med hjälp av en kompilator omvandlar högnivåspråk (ett språk som är enkelt för oss att förstå) till maskinkod (instruktioner som datorn kan förstå). Detta är en tidsödande process, som behöver upprepas vid varje förändring av programkoden. En kompilering ställer höga krav på många olika delar av datorn (bland annat processorn och hårddisken), vilket gör den väl lämpad för vår undersökning.

Det som vi har valt att kompilera är en Linux kärna version 2.2.18, med standard inställningar (www.kernel.org). Kärnan hämtades ner från SUNETs ftp arkiv. Det är inte ovanligt att Linux kärnan används i den här sortens undersökningar. Linux kärnan består av en blandning av stora och små filer vilka kompileras i en oregelbunden ordning, för att sedan länkas ihop till binärfiler, vilket sker seriellt. Kompilerings och länkningsarbetet utförs flera gånger om varandra. Detta innebär att kompileringsprocessen av en Linux kärna har en komplicerad struktur.

Det finns många olika kompilatorer på marknaden, för en mängd olika programspråk. Den kompilator som vi använde oss av är en C kompilator från GNU Compiler Collection, i dagligt tal kallad GCC. Vi kommer även att använda GNU Make för att styra kompileringen. (www.gnu.org)

4.4.3 MSC.Marc

MSC.Marc (version 2000) är en generell applikation som används för att utföra beräkningar enligt FEM (Finita ElementMetoden). FEM består av en mängd komplicerade beräkningar, vilka går att dela in i delproblem, vilket förklaras mer ingående under rubriken 4.4.1 Fluent. (Samuelsson & Wiberg, 1988) MSC har med Marc tagit steget vidare och utvecklat en metod kallad DDM (Domain Decomposition Method). (www.mscsoftware.com) DDM innebär att applikationen delar in matrisen som skall beräknas i flera delar vilka sedan beräknas av var sin nod/processor. Det finns dock vissa problem med DDM och Marc, bland annat att beräkningstiden planar ut vid användande av över tio noder/processorer (Boklund & Larsson, 2001).

Våra mätningar utfördes på en modell vilken vi erhöll av Mikael Eriksson som är doktorand vid Institutionen för Teknik vid Högskolan i Trollhättan/Uddevalla. Modellen skapades som en del i en förstudie till hans doktorandprojekt. Mikaels arbete går ut på att skapa en modell som kan användas till att simulera värmeutveckling och mekanisk påverkan på material vid svetsning med en svetsrobot. Simuleringarna utförs med hjälp av MSC.Marc. (Ericsson, Nylén, Bolmsjö)

4.4.4 POV-Ray

POV-Ray är en så kallad ”Ray Tracing” applikation. Ray Tracing är en metod för att skapa fotorealistiska bilder med hjälp av en dator, vilka ofta tar lång tid att räkna ut. POV-Ray använder matematiska modeller av en miljö för att räkna ut hur ljus faller på olika föremål. Den räknar också ut reflektioner och ljusbrytning på de olika objekten. Processen som omvandlar de matematiska modellerna till bilder kallas för rendering.

Det finns sedan en längre tid en standardiserad bild vid namn Skyvase, som används vid prestandamätningar med POV-Ray. Bilden skall renderas med vissa inställningar, dessa finns beskrivna på PovBench officiella hemsida. (www.haveland.com/povbench) Enligt Tom Figgatt från IBM tog det 1989 mellan 10 och 15 minuter för en VAX minidator att rendera Skyvase (www.cnn.com).

MPIPov är en modifikation av POV-Ray som distribuerar renderingsprocesser över flera datorer med hjälp av MPI. MPIPovs funktion är att dela upp bilden i delar och sedan distribuera delarna så att det är en del per nod, för att noderna skall räkna fram delresultat. Efter att rendering har skett på noderna skickas resultaten tillbaka till huvudnoden för att sammanställas med de andra delresultaten.

Processen att räkna ut alla ljusstrålar är mycket komplicerad. Varje bildpunkt belyses med en eller flera ljusstrålar för att räkna ut vilken färg den aktuella bildpunkten kommer att få. (www.povray.org) Bakom varje bildpunkt ligger därför en komplicerad beräkning. Det finns metoder i POV-Ray för att få simulationen av ljuset ännu bättre. En av dessa metoder är Radiosity som förutsätter en utstrålande reflektion av texturen. En textur beskriver en yta, det vill säga dess färg, reflektion med mera. Det klassiska exemplet för att beskriva Radiosity går ut på att man placerar en vit boll inuti ett rum med blå väggar. Bollen kommer då att färgas blå av ljuset som reflekteras från väggarna. (Foley, 1990)

Fördelen med att använda POV-Ray som applikation är att den har öppen källkod, samt att applikationen är gratis. Programmet har redan modifierats för att passa tillsammans med ett flertal olika meddelandesystem, bland dessa återfinns både MPI och PVM.

5 LABORATORIEMILJÖ

5.1 Hårdvara

För att kunna genomföra våra laboratorieförsök behövde vi en lab-miljö. Denna lab-miljö bestod av tio stycken datorer, som kopplades samman i ett nätverk. Alla installationer gjordes på rackmonterade hårddiskar som numrerades i överensstämmelse med datorernas numrering. När vi inte utförde några laborationer förvarades hårddiskarna inlåsta i ett skåp. Anledningen till detta var att vi ville ha en så kontrollerbar miljö som möjligt.

Datorerna som användes innehöll följande hårdvara:

Processor:	Pentium III 500 MHz
Minne:	128 MB SDRAM
Moderkort:	Asus P3B-F
Nätverkskort	1 st. 3Com 3c905b 10/100
Hårddisk:	IBM 9.1 GB
CD-ROM:	40X
Diskettstation:	1.44 MB
Grafikkort:	Matrox G400 16MB
Ljudkort:	Soundblaster PCI 128

Datorerna var sammankopplade med en Cisco 2900 switch. Cisco 2900 har 12 stycken 10/100 Mbit portar. Hastigheten som datorerna använde mot switchen var 100 Mbit/s. Under laborationerna var varken switchen eller datorerna i kontakt med något annat nätverk, utan var en autonom enhet. De uppgifter om nätutnyttjande och bandbredd som används i analysen, hämtades från switchens inbyggda loggningsfunktioner.

5.2 Operativsystem

Scyld är baserad på linuxdistributionen RedHat version 6.2. På grund av detta valde vi att även basera MOSIX installationen på RedHat version 6.2. (www.redhat.com) RedHat är en av de största och mest namnkunniga Linux distributionerna.

5.3 MOSIX

RedHat distrubutionen som användes till MOSIX laddades ner som en image-fil från Sunets ftp arkiv och brändes på en CD-skiva (ftp.sunet.se). Installationsfilerna för MOSIX hämtades från MOSIX hemsida (www.mosix.org) och en ny Linux kärna laddades ner från The Linux Kernel Archives (www.kernel.org). Anledningen till att en ny Linux kärna laddades ner var att den kärna som följer med RedHat Linux version 6.2 inte är en standard kärna utan innehåller RedHat-specifika förändringar. Genom att ladda ned en ny kärna elimineras risken för att RedHats förändringar påverkar resultaten.

Att installera MOSIX är en smal sak förutsatt att man har grundläggande kunskaper om Linux/Unix system, då dokumentationen är enkel och rak. Det finns dock ett steg i installationen som kräver mer av administratören och det är när Linux kärnan skall kompileras om. För att kunna konfigurera och kompilera kärnan med ett bra resultat krävs att man har ingående information om hårdvaran (drivrutiner) samt kunskaper om hur denna samverkar.

Huvudnoden installerades från en CD-skiva och sedan klonades installationen till de andra hårddiskarna med hjälp av programmet Systemimager (systemimager.sourceforge.net). Detta gav oss nio stycken identiska installationer. Huvudnoden konfigurerades som en server för NFS (Network File System), vilket är ett protokoll för att ge andra datorer tillgång till en lagringsplats på en annan dator (Abrahams & Larson, 1999). En katalog i vilken data som användes av applikationerna kunde hämtas och sparas exporterades till alla noder.

5.4 Scyld

Scyld levereras på en CD-skiva från vilken man kan både installera huvudnoden och starta noderna från. En image-fil av Scyld CD:n laddades ner från Beowulf-Undergound (www.beowulf-underground.org). Scyld laddades inte ner från Scyld:s eget ftp-arkiv på grund av att de inte har lagt ut den för nerladdning. Anledningen till detta är förmodligen att de säljer distributionen genom Linux Central (det är dock inte olagligt att ladda ner den från en tredje parts ftp-arkiv). Av denna image-fil gjordes det sedan flera Scyld Beowulf CD-skivor. En standard installation av Scyld Beowulf gjordes på huvudnoden. Installationen var enkel och krävde endast grundläggande kunskaper i datalogi och datorkommunkation.

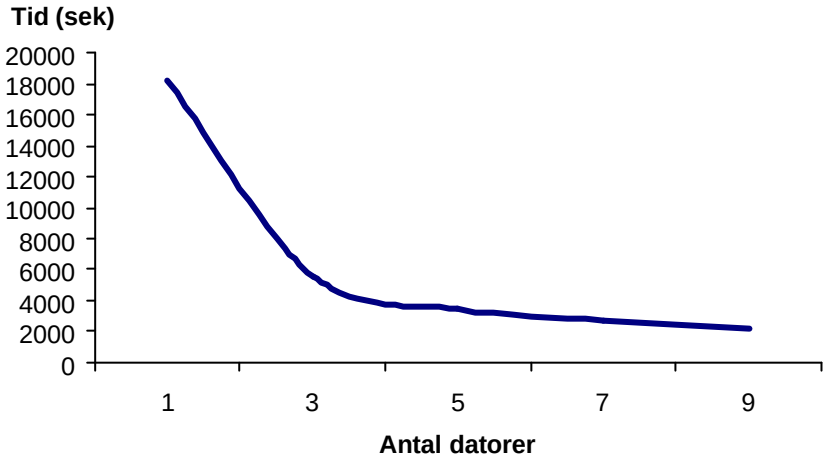
Noderna startades från var sin kopia av Scyld CD:n. Det krävdes en del arbete för att konfigurera huvudnoden och nodernas initialiseringsinformation för att klustret skulle starta och bli operativt. Denna del var väldigt dåligt dokumenterad men tack vare hjälp från anställda på Scyld Computing Corporation lyckades vi få Scyld att fungera ihop med vår hårdvara (beowulf@beowulf.gsfc.nasa.gov).

6 LABORATORIERESULTAT

6.1 Fluent

Applikationen Fluent utförde 100 iterationer (delberäkningar) på tre, fem, sju och nio noder samt på enbart huvudnoden. Tre identiska mätningar utfördes och ett medelvärde av dessa mätningar användes. Värdena som samlades in var tidsåtgång, processor-, minne- samt nätverksutnyttjande.

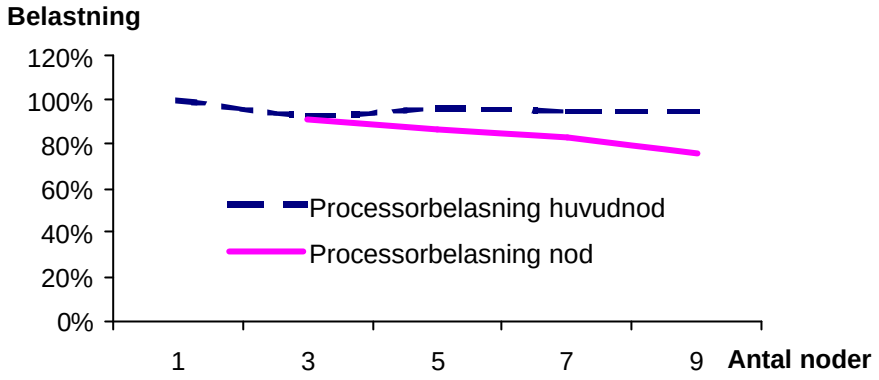
På grund av att Fluent kräver mer funktionalitet än den som finns inkluderad i Scylds nodimplementation gick det inte att utföra några försök med Fluent under Scyld.
Det var dock möjligt att länka om Fluent mot Scylds egna implementation av MPI, BeoMPI (www.scyld.com). Detta kunde bekräftas eftersom det gick att köra Fluent så långt att ett felmeddelade genererades. Det kommer förmodligen att vara möjligt att använda Fluent under kommande Scyld-versioner.



Figur 6.1: Tidskurva för Fluent under MOSIX

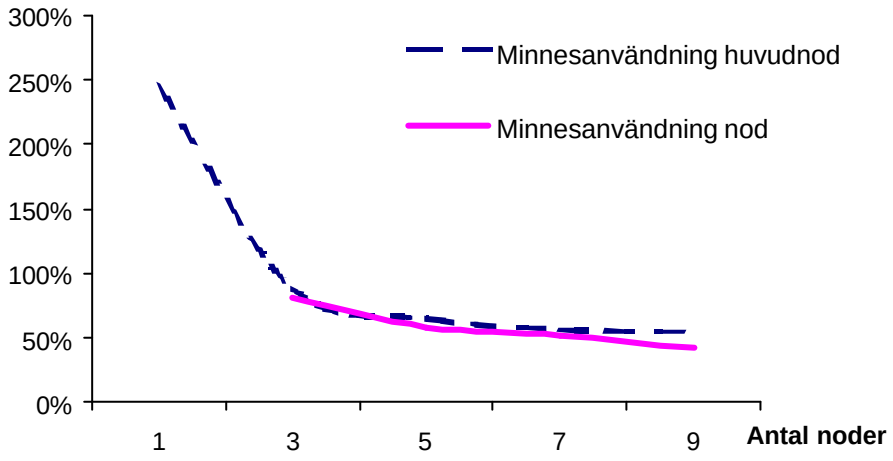
Tidskurvan (Figur 6.1) visar att tidsvinsten per extra nod blir lägre ju fler noder man använder sig av. Detta är ett vanligt fenomen hos parallella implementationer och kan härledas till Amdahls lag. Det beror i det här fallet på den extra information som huvudnoden måste beräkna, bland annat på grund av att den delar upp och distribuerar problemet, samt att nätverksbelastningen ökar med antalet noder.

Processorbelastningskurvan (Figur 6.2) visar att belastningen på huvudnoden är mer eller mindre konstant oavsett hur många noder som ansluts. Processorbelastningen på noderna sjunker dock allteftersom fler noder ansluts. Detta beror på att huvudnoden alltid arbetar under mer eller mindre full belastning, vilket innebär att den inte klarar av att serva noderna med beräkningsinformation i lika stor utsträckning som de begär det. Ju fler noder som ansluts, desto större blir gapet mellan noder och huvudnod. Noderna stod alltså till viss del overksamma och väntade på att erhålla nytt beräkningsmaterial från huvudnoden (eller väntade på att skicka tillbaka resultat från utförda beräkningar). Nätet var dessutom periodvis överbelastat vid användandet av fler än fem noder. Alla dessa faktorer bidrog till den låga processorbelastningen på noderna.



Figur 6.2: Processorbelastning master/nod under MOSIX

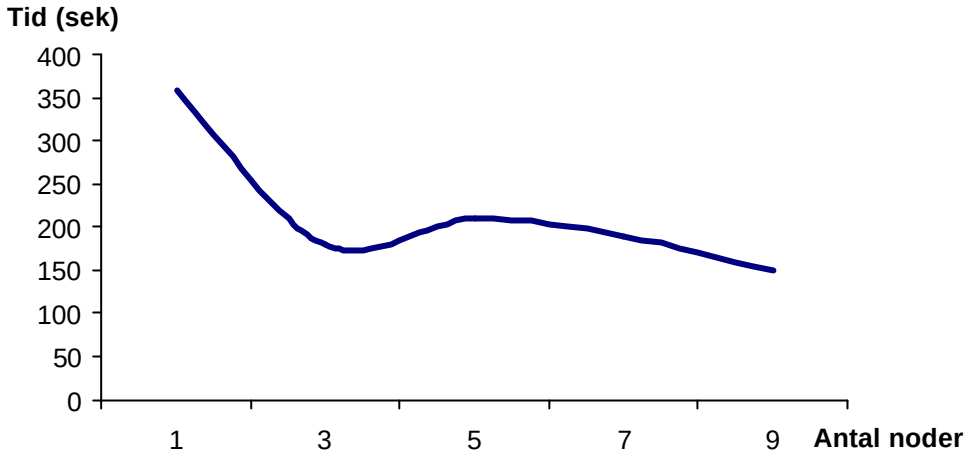
Minnesbelastningskurvan (Figur 6.3) visar att huvudnoden är överbelastad när den sköter alla beräkningar själv (vid minnesanvändning över 100 % används virtuellt minne). När noder används delas problemet upp i fler delar och minnesanvändningen på huvudnoden minskar. Anledningen till att minnesanvändningen på huvudnoden är större än på noderna är att huvudnoden förutom att utföra beräkningar även skickar ut och sammanställer alla beräkningar.



Figur 6.3: Minnesbelastning huvudnod/nod

6.2 Kompilering

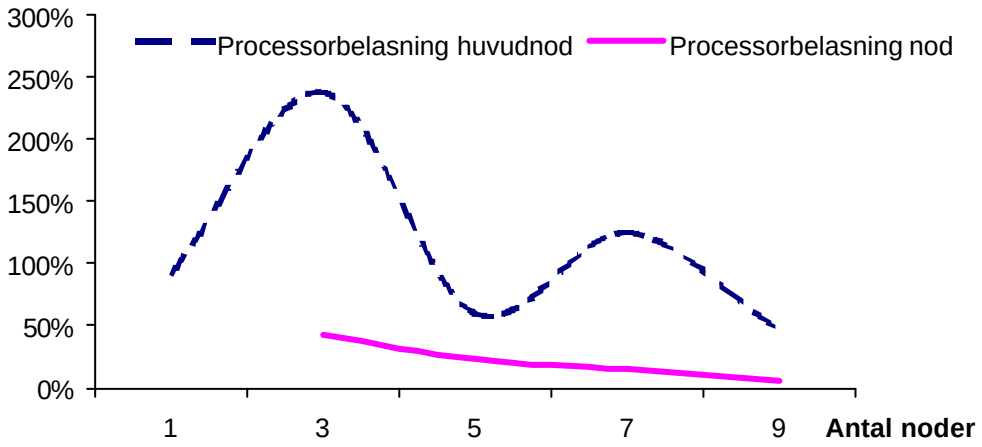
Kompilatorn GCC användes för att kompilera en Linux kärna på tre, fem, sju och nio noder samt på enbart huvudnoden. Även en tredjepartsprogramvara (MPExec/MPMake) hämtades från MOSIX:s hemsida (www.mosix.org). Denna programvara är specialanpassad för att distribuera kompileringsprocesser under MOSIX. Kompileringen utfördes tre gånger och ett medelvärde av dessa mätningar användes. Värdena som samlades in var tidsåtgång, processor-, minne- samt nätverksutnyttjande.



Figur 6.4: Tidsåtgång för kompilering av Linux kärna under MOSIX

Tidskurvan (Figur 6.4) visar att tidsåtgången för kompileringen först minskar för att sedan öka något innan den åter minskar. Detta fenomen kan ha flera olika förklaringar då en kompilering är en avancerad process. Vid en kompilering av en Linux kärna kompileras flera hundra filer av olika storlek, vilka tar allt ifrån några tiondels sekunder till flera sekunder att slutföra.

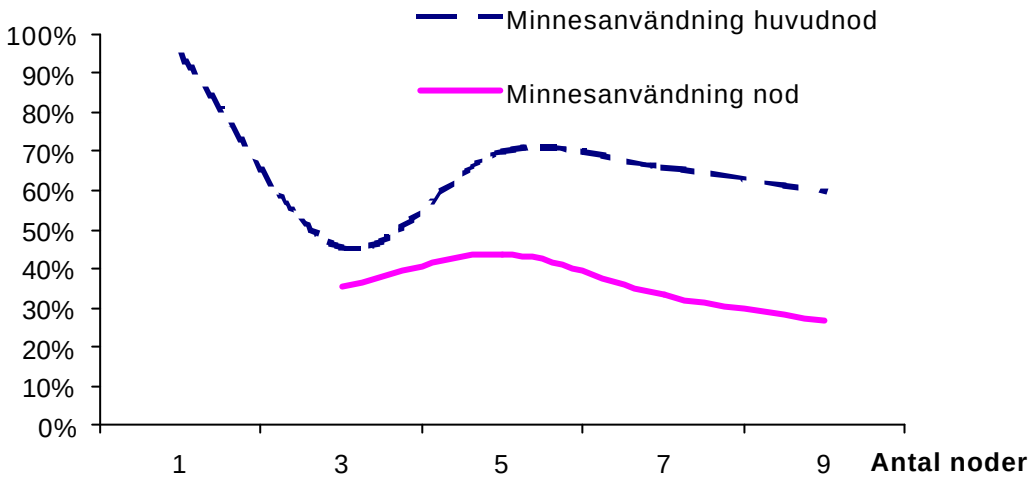
En tänkbar orsak är att vid användandet av tre noder delas de mest resurskrävande filerna ut till noderna, medan samtliga av de mindre krävande filerna kompileras på huvudnoden. Det finns en baslatens inblandat i att kompilera en fil på en annan nod än huvudnoden. Beskrivet i enkla termer är denna latens den extra tidsåtgång som krävs för att paketera, flytta och övervaka kompileringen på en nod. Den extra tidsåtgången är procentuellt större för en mindre fil. När kompileringen sedan sker över fem noder delas även de mindre filerna ut. På grund av latensen tar dessa filer längre tid att kompilera om de delas ut till och utförs på en nod än om huvudnoden utför dem själv, vilket i sin tur leder till en ökad totaltid. Om antalet noder ökas ytterligare ger dock den ökande parallelliseringen tidsvinster även om processen utförs långsammare på en nod än på huvudnoden.



Figur 6.5: Processorbelastning huvudnod/nod under MOSIX

Att mäta processorbelastning (Figur 6.5) och minnesanvändning (Figur 6.6) på ett kompileringsjobb är väldigt svårt eftersom det är ett snittvärde över en viss tidsperiod, som i detta fallet är väldigt kort.

Det måste tas i beaktande att en kompileringsprocess med sin blandning av filstorlekar är en ojämn process som belastar noderna mycket ojämnt. Vi tror därför inte att man kan dra några generella slutsatser från varken processorbelastnings eller minnesanvändningsgraferna. Dessa två följer dock varandra ganska väl vilket torde bero på att mätningarna skedde vid samma tillfälle.



Figur 6.6: Minnesanvändning för Fluent huvudnod/nod under MOSIX

6.3 MSC.Marc

Det gick tyvärr inte att utföra några simuleringar med MSC.Marc, varken under MOSIX eller Scyld. Det gick att starta MSC.Marc under MOSIX och även att utföra en simulering på huvudnoden. När fler noder sedan kopplades in och MOSIX startades, flyttade den inte ut några processer till noderna, utan alla beräkningar utfördes även då på huvudnoden. Anledningen till detta är att MSC.Marc alltid allokerar sitt RAM minne som delat minne (shared memory). MOSIX flyttar inte processer som har allokerat minne som delat, detta på grund av att den i så fall skulle vara tvungen att även flytta de andra processer som kan tänkas vilja läsa respektive skriva till detta minnes segment.

Användandet av delat minne även för fristående processer, som ej delar information med andra processer, bekräftades av Marc Hertlein på MSC Software. MSC Software har tidigare inte sett det som ett problem eftersom de inte har undersökt möjligheten att köra sina produkter varken under MOSIX eller Scyld.

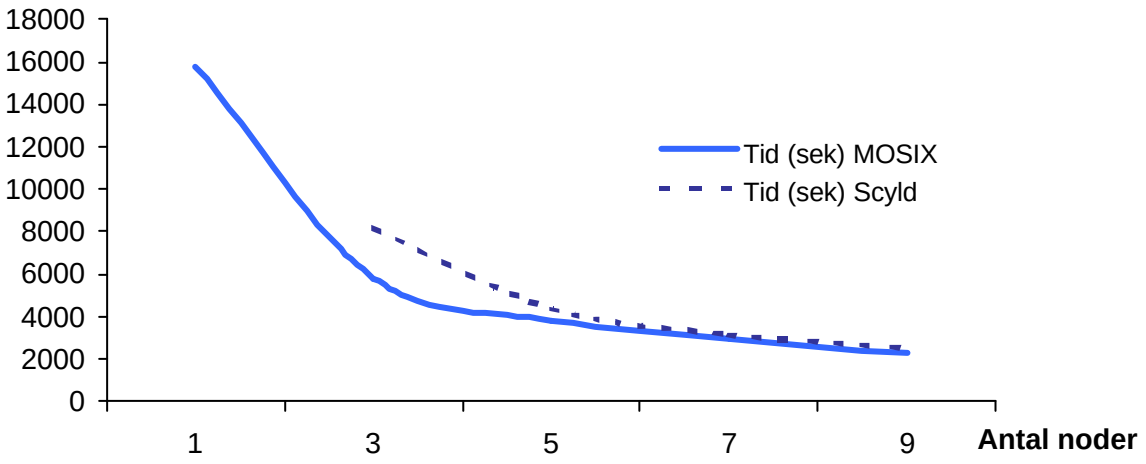
Precis som Fluent kräver MSC.Marc mer funktionalitet än den som finns inkluderad i Scylds nodimplementation. Vi försökte länka om MSC.Marc mot Scylds egen implementation av MPI, BeoMPI (www.scyld.com). Vi vet dock inte om det fungerade eftersom vi inte kunde bekräfta att MSC.Marc startade parallellt. Det enda vi vet säkert är att MSC.Marc startar och sedan avslutas. Det genereras tyvärr inte något felmeddelande som kan förklara varför simuleringen inte går att köra.

6.4 POV-Ray

POV-Ray användes för att rendera en bild (First strike at Pearl Harbor) på tre, fem, sju och nio noder samt enbart på huvudnoden. Tre identiska mätningar utfördes och ett medelvärde av dessa mätningar användes. Värdena som samlades in var tidsåtgång, processor-, minne- samt nätverksutnyttjande.

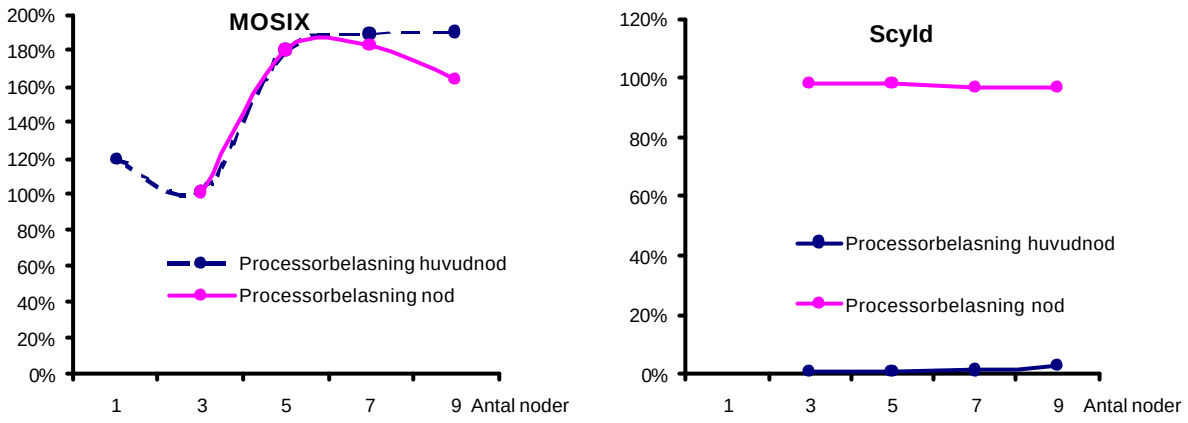
Figur 6.7: First strike at Pearl Harbor av Glenn McCarter (www.povray.org)

Bilden renderas under både Scyld och MOSIX. POV-Ray har ingen egen processfördelning, men den har stöd för både MPI och PVM. Eftersom det finns en MPI implementation för Scyld, BeoMPI, valde vi att använda oss av denna för renderingarna under Scyld. Även för de renderingar som utfördes under MOSIX använde vi oss av en MPI implementation (MPICH) som laddades hem från MPICH hemsida (www-unix.mcs.anl.gov).



Figur 6.8: Tidsåtgång för rendering under MOSIX respektive Scyld

Tidskurvan (Figur 6.8) visar att tidsskillnaden mellan de två processmigreringssystemen är avsevärd vid användandet av tre noder. Denna skillnad utjämnas dock när fler noder används. Skillnaden kan förklaras med att MOSIX även använder huvudnoden till beräkningsarbete medan Scylds huvudnod endast fördelar beräkningar till noderna och sammanställer resultaten. Scyld har dock effektivare processhantering genom att den till skillnad från MOSIX flyttar hela processen till noden istället för att skapa deputy/remote-processer. Detta är anledningen till att tidsåtgången vid användandet av fler än fem noder blir ungefär samma för både MOSIX och Scyld, trots att Scyld har en beräkningsenhet mindre på grund av att huvudnoden inte används till beräkningar.



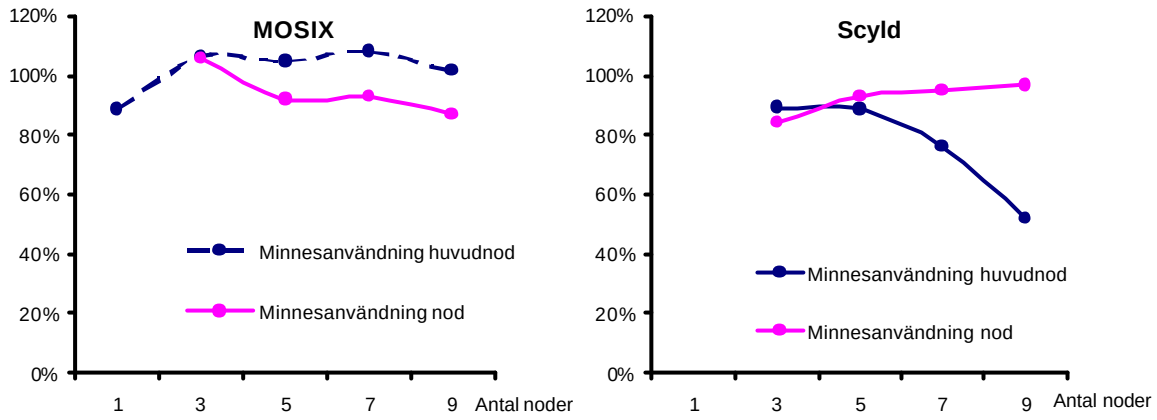
Figur 6.9: Processorbelastning för rendering under MOSIX respektive Scyld

Processorbelastningen skiljer sig åt mellan de två processmigreringssystemen (Figur 6.9). Under MOSIX är processorbelastningen betydligt högre än under Scyld. Detta beror på att en av grundförutsättningarna för att MOSIX skall migrera processer är att huvudnod/noder är överbelastade och därigenom delar ut processer. Processorbelastningskurvan för noderna börjar sjunka vid användandet av fler än fem noder, vilket är ett tecken på att huvudnoden är överbelastad och inte längre klarar av att leverera beräkningsdata i samma takt som noderna efterfrågar. Detta syns även i belastningskurvan för huvudnoden som planar ut vid användandet av fler än fem noder.

Under Scyld är noderna fullbelastade hela tiden medan huvudnoden har extremt låg belastning. Detta beror som tidigare nämnts på att huvudnoden inte utför några beräkningar utan enbart fördelar beräkningar till noderna och sammanställer resultaten. En ytterligare anledning till att processorbelastningen är låg på huvudnoden är att större delen av kommunikationen från huvudnoden hanteras av kärnan, och denna belastning kan ej mätas.

En tydlig skillnad kan ses mellan de två processmigreringssystemen när det gäller minnesanvändning (Figur 6.10). Vid användning av MOSIX sjunker minnesanvändningen vid användandet av fler noder vilket beror på att den totala problemstorleken delas mellan fler noder. Skillnaden i minnesanvändning mellan huvudnod och nod kan härledas till det extra minne som krävs för att dela upp och distribuera problemet till noderna samt sammanställa resultatet.

Vid användning av Scyld har minnesanvändningskurvan en annan karaktär (Figur 6.10). Här ökar minnesanvändningen hos noderna upp till närmare 100 % medan den hos huvudnoden sjunker kraftigt vid användandet av fler än fem noder. Anledningen till den minskade minnesanvändningen hos huvudnoden är att den vid ökat antal noder kan distribuera en större del av problemet omedelbart och undviker därmed att lagra upp delar av problemet i det egna minnet.



Figur 6.10: Minnesanvändning för rendering under MOSIX respektive Scyld

7 RESULTATANALYS

7.1 Resultat och metodkritik

Med reliabilitet menas att de resultat som kommer fram är framtagna på ett tillförlitligt sätt, och att det inte finns slumpmässiga fel som påverkar resultaten. Ofta är reliabilitet även lika med reproducerbarhet, det vill säga försöken skall gå att återupprepa. (Patel & Davidson, 1994) För att försäkra oss om en hög reliabilitet kommer vi i undersökningen se till att de laboratorietester som utförs dokumenteras och att slumpfaktorer i utförandet därigenom elimineras.

Samma mätsystem har använts för alla laborationer. Detta system går ut på att mäta processorbelastningen var tionde sekund. Det är dock så att antalet mätningar under en 18 245 sekunders Fluent modell ger ett relativt rättvisande värde för processorbelastningen (1 824 mätvärden), medan antalet mätvärden på en 150 till 358 sekunders kompilering endast är 15 till 35 stycken. Detta kan ha påverkat våra resultat.

Vi anser att vår undersökning har god validitet, det vill säga vi har undersökt det som vi avsåg att undersöka (Patel & Davidson, 1994). Detta baserat på att det som vi är intresserade av att mäta är kvantifierbara värden, som har hämtats direkt från Linux kärnan eller från nätverkets knytpunkt, det vill säga switchen. Undantaget är mätandet av den tid som det tar att utföra en beräkning och det som vi mäter där är just tidsåtgången. Denna har mätts antingen av applikationen själv eller genom ett till applikationen knutet startscript.

7.2 Installation

Installationen av båda processmigreringssystemen gick relativt smärtfritt. För Scyld krävs inte någon expertkunskap utan en person med grundläggande datorkunskaper kan med hjälp av installationsanvisningarna som följer med sätta upp ett kluster. MOSIX däremot kräver djupare kunskaper. Man behöver ha kännedom om UNIX, datornätverk samt veta hur man kompilerar om en Linux kärna. Man behöver även ha en större kännedom om datorernas hårdvara.

Ur kostnadssynpunkt har Scyld och MOSIX sina respektive fördelar. Scyld kan köras på noder med mindre/billigare hårdvara då exempelvis hårddiskar ej behövs. MOSIX kan däremot köras som ett icke dedicerat kluster och därmed använda befintlig hårdvara, genom att använda användarnas datorer som noder i ett kluster.

7.3 Användbarhet

MOSIX och Scyld är de två största processmigreringssystemen som finns till Linux. Likheterna mellan dem är att de används för att skapa klustersystem och hanterar detta på processnivå. Processen startas på en dator och flyttas sedan till en annan dator. Sättet som de genomför detta på är helt skilda från varandra. MOSIX använder sig av speciella algoritmer som flyttar processerna från en dator till en annan när datorn blir överbelastad. Scyld däremot förlitar sig på att programmerarna i förväg bestämmer hur processerna skall placeras, precis på samma sätt som när man programmerar mot ett meddelandesystem (MPI, PVM). Detta innebär att Scyld kräver specialskriven programvara, medan MOSIX klarar av att parallellisera mer eller mindre alla program. MOSIX klarar dock inte av att bryta ner en process i flera delar och distribuera dem över flera datorer, men en programvara som fungerar på en SMP dator och inte använder sig av delat minne kan köras parallellt under MOSIX, dock inte alltid med ett fördelaktigt resultat.

Scyld har nackdelen att nodimplementationerna är ”tunna” och inte innehåller all den funktionalitet som en del programvaror kräver, vilket dock även är en fördel eftersom de då har mer beräkningskraft över, samt att installationen är enklare. Frånvaron av funktionalitet påverkade undersökningen till den grad att det endast gick att utföra en av laborationerna på Scyld klustret. MSC.Marc gick inte att köra på MOSIX, vilket berodde på att MSC.Marc allokerar sitt minne som delat minne trots att den inte använder sig av den funktionaliteten. MOSIX har inte stöd för delat minne mellan noder och flyttar därför inte processer med delat minne till andra noder.

MOSIX fungerar lika bra över heterogen som homogen hårdvara, och är byggd för att fungera både som ett dedicerat och ickededicerat kluster. De algoritmer som placerar ut processer på andra noder tar hänsyn till om en av noderna arbetar med någonting annat samt hur kraftfull noden är. Scyld däremot fungerar endast som ett dedicerat kluster och hanterar inte heterogen hårdvara då den förutsätter att alla noder är lika.

7.4 Funktionalitet

Utifrån de laboratorieförsök som vi har utfört kan man dra slutsatsen att Scyld utnyttjar processorn till en högre grad än MOSIX. En rendering är en processor- och minneskrävande operation, men då renderingen under Scyld utnyttjade processorn till nästan 100 %, tror vi att minnesbelastningen är sekundär. Renderingsprocessen skrev inte någonting till hårddisken och hade låg nätverksbelastning. Scyld tangerade nästan MOSIX totaltid för renderings processer över fler än fem noder trots att den hade en processor mindre, på grund av att huvudnoden inte utförde några beräkningar. Scylds tid för rendering över tre datorer, där den använde två noder för beräkningar är anmärkningsvärt bra. Tidsåtgången var nästan exakt hälften av vad det tog för MOSIX att endast rendera på huvudnoden. Fluent är ett program som till största delen är processor beroende (Boklund, Larsson, 2000). MOSIX visade upp ett nästan konstant 100 % processorutnyttjande på huvudnoden, medan nodernas processorbelastning sjönk när antalet noder ökade.

När en Scyld nod startas skapar den en RAM-disk på vilken operativsystemets data lagras. Denna RAM-disk är lika stor oavsett hur mycket fysiskt minne som finns i noden. Scyld-noderna har därför mindre minne tillgängligt för beräkningsprocesser än MOSIX noderna. Scyld tar ingen hänsyn till mängden ledigt minne som en nod har när den placerar ut en process. Detta är ytterligare en anledning till att Scyld inte är lämpat för att köras som ett heterogent kluster. MOSIX däremot mäter mängden använt RAM minne för en process och ser sedan om det finns någon nod som har nog mycket RAM för att täcka behovet. Om det finns en nod med tillräcklig RAM kapacitet och den kör en annan process som kräver mindre RAM, kan MOSIX flytta den processen till en annan nod. MOSIX resursdelningsalgoritmer försöker hela tiden optimera klustret, vilket lönar sig vid längre beräkningar men inte är lika effektivt vid körning av många korta. Detta framgår tydligt vid tolkning av värdena som erhölls vid kompileringen.

All skrivning av processer till hårddisk under MOSIX sker via deputy processen, detta leder till att systemanropen skickas över nätverket och att skrivningen sedan utförs på den unika hemmanoden. Scyld fungerar ungefär likadant men här skriver processen direkt till en NFS-volym, vilken är utdelad från huvudnoden eller en extern filserver. MOSIX gör även så att om en process har mycket I/O trafik låter den den processen stanna på huvudnoden och flyttar ut andra processer. MOSIX noderna har även möjlighet att använda sig av virtuellt minne, medan Scyld noderna i såfall måste sköta även denna trafik över nätverket. Scylds filosofi är att man skall köpa mer RAM minne om minnesbehovet är större än andelen tillgängligt minne. Man måste ha i åtanke att Scyld är ett kommersiellt initiativ medan MOSIX är ett forskningsprojekt. Det är dock så att processmigreringssystem i allmänhet inte är anpassade för att hantera processer som har ett stort behov av hårddiskkapacitet eller hårddiskprestanda.

Under nästan alla försök med fler än fem noder var nätverket periodvis överbelastat. Beroende på antalet noder och problemets uppbyggnad ställs olika krav på nätverket. I vissa fall kan det vara fullt tillräckligt med ett 100 Mbit switchat nätverk, medan det i andra fall kan krävas kraftfullare nät än vad budgeten tillåter. Förmodligen är det så att processmigrerings-system har ett stort behov av nätverkshastighet och bandbredd. En variant för att lösa detta är att använda sig av antingen ett snabbare nätverk eller en annan nätverkstopologi än ett bussnät, till exempel ett fullständigt förbundet nätverk (det vill säga alla noder har en dedicerad förbindelse till alla andra noder). (Boklund, 2001) De krav som kommer att ställas på nätverket bör utredas noggrant innan man konfigurerar ett kluster.

7.5 Framtida klustersystem

Versionen av Scyld som vi använde oss av, 27bz6, var den första publika versionen. Den innehåller en hel del brister som förmodligen kommer att rättas till i senare versioner. Problem som vi hade med Fluent var att Fluent inte kunde utföra ett systemanrop på grund av att det inte stöddes av noderna. Stöd för detta systemanrop finns med i Scyld:s senaste version 27bz8. Vi vet inte om Fluent nu fungerar under Scyld, det finns dock ett hinder mindre.

Amnon Barak och hans forskningsgrupp som arbetar med MOSIX har flera projekt på gång. Bland dessa återfinns projekt som syftar åt att låta alla noder skriva till ett parallellt filsystem som är distribuerat mellan noderna, att införa stöd för distribuerat delat minne samt nätverks RAM där man flyttar processen till data och inte tvärtom. Dessa projekt kommer förmodligen att göra det möjligt för MOSIX att på ett bättre sätt hantera I/O intensiva programvaror.

8 SLUTSATSER

Är processmigreringssystemen MOSIX och Scyld användbara? MOSIX klarar idag av att hantera merparten av de programvaror som vi använde oss av, det finns dock vissa program som inte går att köra alls. Scyld klarade däremot inte av att hantera tre av de fyra programvaror som vi testade och har en hel del brister. Båda har potential att bli väl fungerande processmigreringssystem men i dagsläget är det bara MOSIX som är så pass förfinat att det generellt sett är användbart.

Båda processmigreringssystemen har generellt sett samma styrkor och svagheter. De har dock sina egenheter. MOSIX är betydligt svårare att installera och underhålla och har inte heller riktigt samma kraft när det gäller ren beräkningsprestanda. Scyld däremot är lättare att installera och har lägre krav på hårdvaran, men kan dock endast köras dedicerat i en homogen miljö.

MOSIX är bättre på att hantera minneskrävande uppgifter genom användande av lokalt virtuellt minne samt att den placerar processen på den nod som lämpar sig bäst för uppgiften. Båda processmigreringssystemen hanterar nätverket på olika sätt. MOSIX har ett högre bandbreddsutnyttjande på grund av den information som skickas mellan dess remote och deputy processer, medan Scyld ställer högre krav när det handlar om mer sekundärminnesintensiva operationer.

System	MOSIX	Scyld
Processorintensiva		X
Minnesintensiva	X	
Sekundärminnesintensiva	X	
Nätverksintensiva	X?	X?

Figur: 8.1 Vilka applikationstyper lämpar sig MOSIX respektive Scyld bäst för?

Det är inte endast valet av applikationer som är allenarådande vid valet av processmigreringssystem och klusterarkitektur, utan även kostnaden för inköp, uppgraderingar och drift. Man kan tänka sig ett scenario där pengarna som skulle investerats i en superdator kan användas till att uppgradera användarnas arbetsstationer samt arbetsplatsens nätverk. Dessa arbetsstationer kan sedan användas som ett ickededicerat heterogent kluster under dagen och ett homogent dedicerat kluster under natten. Nackdelen är dock att den erhållna prestandan under dagtid blir något lägre. Hur stor skillnaden är beror på vilka applikationer som användarna kör, både på klustret och på sina arbetsstationer.

8.1 Förslag till vidare studier

Det vore intressant att installera och utvärdera användbarhet och prestandaskillnader mellan ett dedicerat och ett ickededicerat klustersystem. Detta för att utreda om den kapacitet som en kontorsapplikations användare använder sig av förlänger ledtiden för en beräkning nämnvärt. Är det intressant att introducera kontorsmaskiner som beräkningsnoder i ett klustersystem?

Microsofts produkter (Windows serien) har under det senaste året blivit stabilare och utvecklas från ett rent skrivbordsoperativsystem mot att bli ett alternativ för beräkningsdatorer. Det vore intressant att undersöka om det är möjligt att konstruera ett klustersystem bestående av datorer som använder sig av Windows som operativsystem. Det finns flera infallsvinklar: Är det en möjlig konstruktion? Hur är dessa system kostnadsmässigt när det gäller till exempel underhåll? Skiljer sig prestandan mellan de olika operativsystemen? Hur tillförlitliga/stabila är systemen?

Konstruktion av ett heterogent kluster med stor spridning i hårdvara hade varit intressant att undersöka. Detta för att utreda vad det har för fördelar/nackdelar. Klustret borde bestå av en bred blandning av datorer, från 486'or till Alpha eller Itanium 64 bitars processorer.

Det vore intressant att med hjälp av klusterteknik och snabba nätverk låta datorer på en arbetsplats samarbeta. Detta genom att datorerna hjälps åt att utföra alla uppgifter. När till exempel en användare vill starta ett program så kommer övriga användares datorer att hjälpa den datorn, vilket innebär att uppstarten av programmet går fortare. Man skulle då kunna förlänga livslängden på datorerna eftersom man får bra prestanda även med äldre datorer.

9 REFERENSER

9.1 Litteratur

Abrahams, P.W., & Larson, B.R. (1999). *Unix for the Impatient.* (2:a uppl.). USA: Addison Wesley Longman, Inc.

Backman, J. (1998). *Rapporter och uppsatser*. Lund: Studentlitteratur.

Barak, A., Guday, S., & Wheeler, R G. (1993). *The MOSIX Distributed Operating System*. Heidelberg: Springer-Verlag.

Barak, A., La'adan, O., & Shiloh, A. (1999). *Scalable Cluster Computing with MOSIX for Linux*. Jerusalem: The Hebrew University of Jerusalem Israel

Beowulf. (2000). *Beowulf: A New Translation*. Faber

Boklund, A. (2001). *Home Clusters.* ;login magazine, August

Boklund, A., & Larsson, F. (2000). *Parallelldatorsystem, Kluster – Morgondagens superdatorer*. (C-uppsats). Uddevalla: Högskolan Trollhättan Uddevalla, Institutionen för Ekonomi och ADB

Boklund, A., & Larsson, F. (2001). *The Kluster-II Project*: Poster presented on IEEE CCGRID International Symposium on Cluster Computing and the Grid, Brisbane, Australia

Brown, R. G. (2001), *Engineering a Beowulf-style compute cluster.* Duke University

Buyya, Rajkumar (1999a), *High Performance Cluster Computing: Architecture and Systems, Volume 1*. Upper Sadle river: Prentice Hall.

Buyya, Rajkumar (1999b), *High Performance Cluster Computing: Programming and Applications, Volume 2*. Upper Sadle river: Prentice Hall.

Ericsson, M., Nylén, P., & Bolmsjö, G. (2001). *Three-dimensional simulation of robot path and heat transfer of a tig-welded part with complex geometry.* Trollhättan: University of Trollhättan/Uddevalla

Foley, J.D. (1990). *Computer Graphics: Principles and practice.* (2nd ed.) Reading Massachusetts: Addison-Wesley

Hawick, K.A., Grove, D.A., Coddington, P.D., & Buntine, M.A. (2000). *Commodity Cluster Computing for Computational Chemistry.* Adelaide: University of Adelaide Australia, Department of Computer Science

Hendriks, E. (1999). *BPROC: A distributed PID space for Beowulf clusters.* Center of Excellence in Space Data and Information Sciences. NASA

Hunt, C. (1998). *TCP/IP Network Administration.* (2nd ed.). Sebastopol: O'Reilly & Associates

Patel. R., & Davidson. B. (1994). *Forskningsmetodikens grunder.* (2:a uppl.) Lund: Studentlitteratur.

Samuelsson, A., & Wiberg, N.-E. (1988). *Finita elementmetodens grunder*. Lund: Studentlitteratur

Scyld Computing Corporation (2001). *Scyld Beowulf Reference Manual*. Version 1.03. Scyld Computing Corporation

Spector, D.H.M. (2000). *Building Linux Clusters*. Sebastopol: O'Reilly & Associates

Widerheim-Paul, F., & Eriksson L T. (1991). *Att utreda, forska och rapportera*. (4:e uppl.) Malmö: Liber-Hermods förlag.

Wilson, G. (1995). *Practical parallel programming.* Camebridge: MIT Press.

9.2 Elektroniska källor

Beowulf Project at CESDIS
http://beowulf.gsfc.nasa.gov/ (2001-01-13)

Beowulf underground hemsida
http://www.beowulf-underground.org/ (2001-01-13)

Beowulf-mailing list beowulf@beowulf.gsfc.nasa.gov
http://www.beowulf.org/listarchives/beowulf/

Cisco Network Academy
http://cisco.netacad.net (2001-08-15)

Computer Science & Mathematics Division, Oak Ridge National Laboratory
http://www.epm.ornl.gov (2001-02-14)

Cnn.com
http://www.cnn.com/TECH/computing/9903/16/super.idg/ (2001-02-15)

The GNU Project
http://www.gnu.org (2001-03-18)

The Linux Kernel Archive
http://www.kernel.org (2001-03-20)

Linux Documentation Project
http://www.linuxdoc.org (2001-03-17)

MOSIX hemsida
http://www.mosix.org (2001-01-04)

MPICH hemsida
http://www-unix.mcs.anl.gov/mpi/mpich/ (2001-02-17)

MSC.Software hemsida
http://www.mscsoftware.com (2001-01-21)

Myricom Inc. Hemsida om Myrinet
http://www.myrinet.com (2001-02-08)

The Portland Group hemsida
http://www.pgroup.com/ (2001-02-27)

Povbenchs hemsida
http://www.haveland.com/povbench (2001-02-25)

POV-Rays hemsida
http://www.povray.org/ (2001-02-03)

Redhats hemsida
http://www.redhat.com (2001-01-12)

Scyld computing corporation hemsida
http://www.scyld.com (2001-02-01)

Sunet:s ftp arkiv
ftp://ftp.sunet.se (2001-01-15)

SystemImagers hemsida
http://systemimager.sourceforge.net (2001-02-13)

Tom’s Hardware Guide
http://www.tomshardware.com (2001-03-01)

Bilaga 1: Begreppsförklaring

Begrepp	Förklaring	Sida
Dedicerat	Används enbart för en uppgift åt gången	13
Exekveringstid	Den tid som det tar att utföra en uppgift	-
Ickededicerat	Kan utföra flera uppgifter samtidigt	13
Heterogen	Alla noder är inte likadana	13
Homogen	Alla noder är identiska	13
Huvudnod	Den nod som styr klustrets arbete	9
Kluster	Sammankopplade datorer som arbetar med att lösa samma problem	13
Lastbalansering	Fördelning av arbete mellan noder så att alla belastas lika mycket	13
MPP	Massively Parallell Processors, En MPP är en stor parallelldator som oftast består av hundratals av beräkningsenheter.	12
Nod, noder	En av datorerna i ett kluster	13
Processmigrering	Att flytta redan startade processer från en dator till en annan.	18
SMP	Symmetric Multi Processing. Innebär att två eller flera processorer är monterade på samma moderkort.	12