NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET

FAKULTET FOR INFORMASJONSTEKNOLOGI, MATEMATIKK OG ELEKTROTEKNIKK

HOVEDOPPGAVE

Kandidatens navn: Erlend Wathne Oftedal

Fag: Datateknikk

Oppgavens tittel (norsk): InfiniBand i parallelle databasesystemer

Oppgavens tittel (engelsk): InfiniBand in parallell database systems

Oppgavens tekst:

Bruk av IP-protokollen fører til høy CPU-bruk og begrenser dermed ytelsen til et parallelt shared-nothing databasesystem. Kandidaten skal gi en oversikt over maskinvare som kan øke kommunikasjonsytelsen mellom nodene og undersøke om InfiniBand og uDAPL kan brukes for å oppnå høyere ytelse i et slikt databasesystem.

Oppgaven gitt:	20. januar 2004
Besvarelsen leveres innen:	15. juni 2004
Besvarelsen levert:	15. juni 2004
Utført ved:	Institutt for datateknikk og informasjonsvitenskap
Veiledere:	Svein-Olaf Hvasshovd og Øystein Torbjørnsen

Trondheim, 15. juni 2004

Svein-Olaf Hvasshovd
Faglærer

Forord

Denne rapporten er resultatet av avsluttende diplomoppgave for 5-årig sivilingeniørutdanning innen datateknikk ved Norges Teknisk-Naturvitenskapelige Universitet (NTNU) våren 2004. Rapporten er et svar på oppgaveteksten:

> *Bruk av IP-protokollen fører til høy CPU-bruk og begrenser dermed ytelsen til et parallelt shared-nothing databasesystem. Kandidaten skal gi en oversikt over maskinvare som kan øke kommunikasjonsytelsen mellom nodene og undersøke om InfiniBand og uDAPL kan brukes for å oppnå høyere ytelse i et slikt databasesystem.*

Ved innlevering var også en CD med kode vedlagt rapporten.

Jeg ønsker å takke mine veiledere Svein-Olaf Hvasshovd ved NTNU og Øystein Torbjørnsen ved Sun Microsystems for god veiledning og verdifulle diskusjoner. Jeg vil også takke Olav Sandstaa, Sherman Pun, Prashant Ramarao og de andre ved Sun Microsystems som har tatt seg tid til å svare på spørsmål og hjelpe til ved løsing av problemstillinger. Til slutt vil jeg takke Sun Microsystems for lån av maskinvare.

Trondheim, 15. juni 2004

Erlend Oftedal

Sammendrag

I et parallelt shared-nothing databasesystem er ytelsen svært avhengig av effektiv kommunikasjon. Mange parallelle databasesystemer benytter IP-protokollen og Ethernet for kommunikasjon mellom nodene, men det viser seg at denne protokollen medfører at prosessor bruker mye tid på oppgaver knyttet til meldingssending og mottak.

Det er gitt en beskrivelse av utvalgt alternativ teknologi for kommunikasjon både internt i datamaskinen og eksternt mot andre datamaskiner. InfiniBand er en av disse teknologiene, og denne ses på som et godt alternativ til Ethernet.

Det er gjort en kartlegging av hvorfor IP-protokollen medfører høy CPU-bruk, og det viser seg at dette skyldes avbrudd, checksum-beregninger og minne-til-minne-kopiering. *User Direct Access Programming Library*, eller uDAPL, er en API for kommunikasjon over InfiniBand. Gjennom denne kan minne-til-minne-kopiering unngås, og checksum-beregning i hardware kombinert med bruk av DMA gjør at uDAPL og InfiniBand skal gi en reduksjon i CPU-bruk.

En ytelsestest og en begrenset implementasjon i Sun HADB ble utført for å sammenligne uDAPL og 4X InfiniBand mot UDP over Gigabit Ethernet. Resultatene viser at uDAPL og InfiniBand kan øke gjennomstrømningen og senke responstiden til et parallelt shared-nothing databasesystem.

Innholdsfortegnelse

Kapittel 1

Innledning

Kommunikasjon legger begrensninger på ytelsen til et parallelt databasesystem. Evalueringer har vist at dagens bruk av IP-protokollen og Ethernet-hardware medfører at datamaskinen bruker mye CPU-tid på håndtering av kommunikasjon når meldingsutvekslingen blir høy.

Det vurderes i denne oppgaven om bruk av InfiniBand-hardware kan redusere CPU-forbruket og dermed øke et parallelt databasesystems ytelse.

Rapporten gir gjennom kapittel 2 en innføring i oppbygning og kommunikasjonsbehov for et parallelt databasesystem. Kapittel 3 introduserer noen begreper knyttet til kommunikasjon både internt i en datamaskin og mot andre datamaskiner. Kapittel 4 og 5 gir en "State of the Art" for alternativ teknologi for intern og ekstern kommunikasjon som kan gi bedre kommunikasjonsytelse mellom noder. En introduksjon til InfiniBand gis i kapittel 6, mens kapittel 7 beskriver problemer knyttet til bruk av IP-protokollen og viser hvordan InfiniBand løser disse problemene. Kapittel 8 introduserer uDAPL som en API for kommunikasjon over InfiniBand. Ytelsesmålingen i kapittel 9 sammenligner CPU-forbruk og overføringsrate ved bruk av IP-protokollen kontra uDAPL over InfiniBand. Kapittel 10 gjengir design og resultater fra en begrenset innføring av uDAPL og InfiniBand i det parallelle databasesystemet Sun HADB.

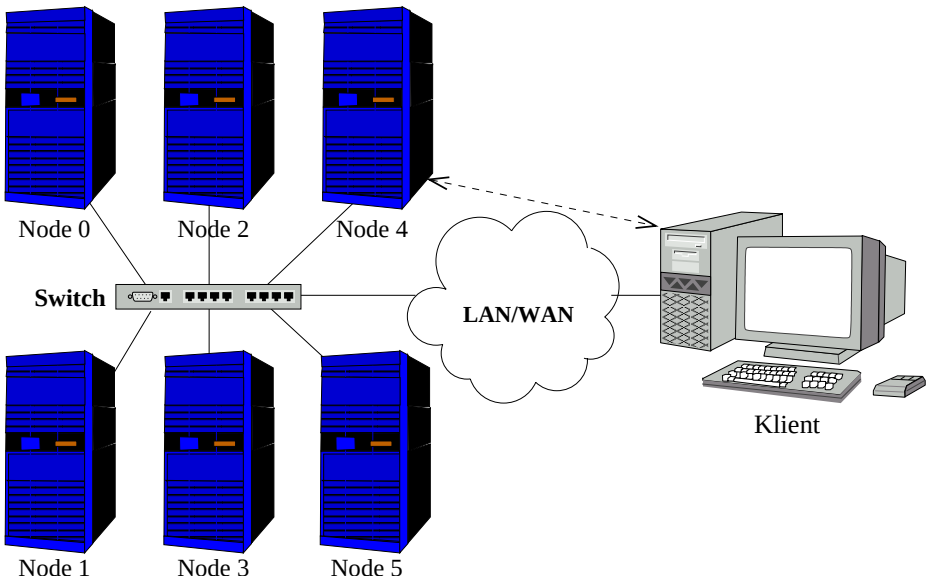
Kapittel 2

Bakgrunn

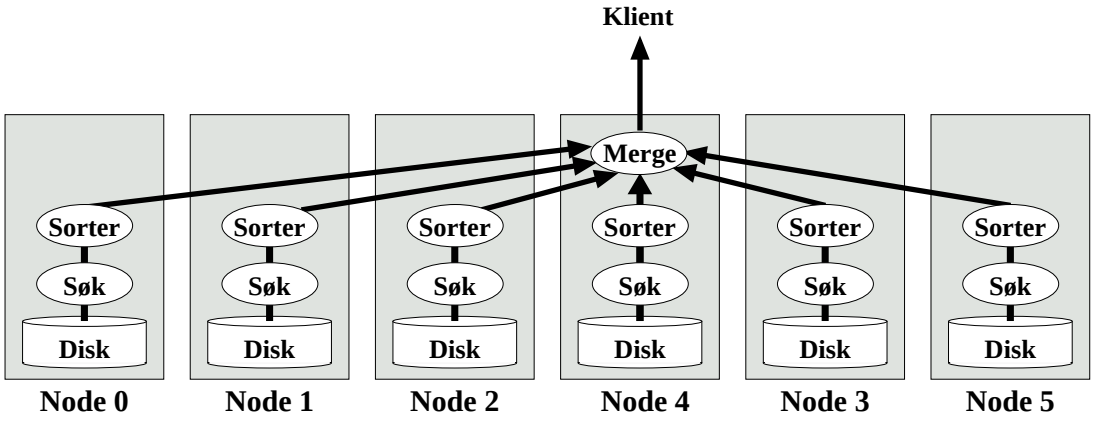
Dette kapittelet beskriver parallelle databasesystemer og kommunikasjon innenfor slike systemer.

2.1 Parallelle databasesystemer

Et parallelt databasesystem består av en gruppe datamaskiner, ofte kalt noder, som samarbeider om å løse oppgavene som påtrykkes systemet. Dette er en mer kostnadseffektiv løsning enn superdatamaskiner både med tanke på førstegangs innkjøp og senere oppgraderinger og utvidelser.



Figur 2.1: Organisering av en parallell shared-nothing database



Figur 2.2: Sortering utført i et parallelt databasesystem

Parallelle databasesystemer kan deles i tre hovedgrupper etter hvilke ressurser som deles mellom nodene. I en *shared-memory*-arkitektur deler nodene primærminne, men har egen disk og prosessor. I en *shared-disk*-arkitektur deler nodene disk, men har eget primærminne og egen prosessor. Denne oppgaven fokuserer på *shared-nothing*-arkitektur der nodene verken deler primærminne eller disk. Figur 2.1 viser oppbygningen av et typisk *shared-nothing*-system. Sun HADB (tidligere Clustra) og IBM DB2 Parallel Edition er eksempler på shared-nothing-systemer.

2.2 Transaksjonsutførelse

En klient kan koble seg til en hvilken som helst node i systemet. Hvilken node klienten kobler seg til kan velges ved at klienten velger tilfeldig blant nodenes IP-adresser, eller en lastbalanserende proxy kan fordele klientforespørsler på nodene. Noden som mottar en transaksjon, er ansvarlig for koordinering og retur av resultat. Den fordeler deloppgaver på nodene i systemet, og samordner delresultater til et ferdig resultat. I et parallelt databasesystem der tabeller fordeles tuppelvis på nodene (horisontal fordeling [TO03]), vil eksempelvis nodene utføre spørringer på sine data og returnere tupler til koordinatornoden. Figur 2.2 viser utførelse av relasjonsalgebra i et parallelt databasesystem.

En viktig egenskap ved en transaksjon er at den er atomisk, det vi si den er enten utført i sin helhet eller ikke utført i det hele tatt. Dersom en transaksjon skal utføres, må den dermed utføres på alle noder den berører. For å bestemme om transaksjonen kan utføres eller ikke, brukes 2PC-protokollen (2 Phase Commit) [GR93] eller varianter av denne. 2PC utføres, som navnet tilsier, i to faser. Protokollen styres av koordinator, som i første fase sjekker om transaksjonen kan utføres hos alle deltakende noder. Kan den det, comittes transaksjonen. Dersom en av nodene ikke kan utføre transaksjonen, må transaksjonen avbry-

System type	Unavailability (minutes/year)	Availability (in percent)	Availability Class
Unmanaged	50.000	90	1
Managed	5.000	99	2
Well-managed	500	99,9	3
Fault-tolerant	50	99,99	4
High-availability	5	99,999	5
Very-high-availability	0,5	99,9999	6
Ultra-high-availability	0,05	99,99999	7

Tabell 2.1: Tilgjengelighetsklasser

tes. I andre fase sender koordinator ut resultatet (commit/abort) til deltakende noder.

2.3 Tilgjengelighet

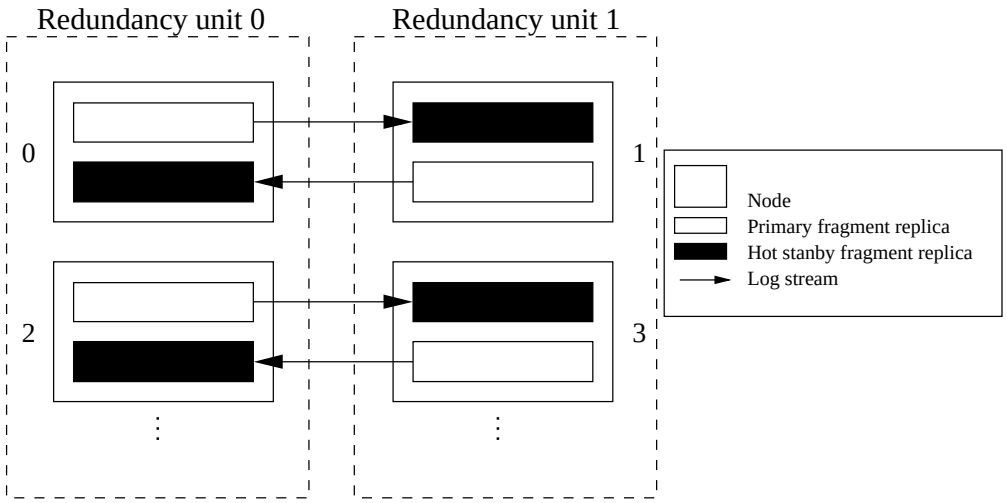
Høytilgjengelige systemer kan klassifiseres etter hvor lang nedetid de maksimalt tillates å ha i løpet av et år [GS91]. Tabell 2.1 viser en slik inndeling. Tilgjengelighetsklassene kan også angis i prosent, og denne faktoren kan regnes ut på bakgrunn av to faktorer spesifisert for systemet. MTTF, eller *mean time between failure*, anslår gjennomsnittlig tid mellom feil, mens MTTR, eller *mean time to repair*, anslår hvor lang tid det gjennomsnittlig tar å reparere en feil. Tilgjengelighet kan da regnes ut som følger:

$$Tilgjengelighet = \frac{MTTF}{MTTF + MTTR} \tag{2.1}$$

Denne formelen tilsier at dersom det tar svært kort tid å reparere en feil, vil systemet få høy tilgjengelighet. I et parallelt databasesystem kan denne tankegangen utnyttes ved at flere kopier av data er tilgjengelig ved normal bruk.

Et eksempel på en systemarkitektur som benytter dette, er beskrevet i [BH01]. I denne arkitekturen er hver node hovedansvarlig for sitt fragment, eller del, av datamengden. I tillegg er to og to noder backup for hverandres fragmenter. Eksempelvis er node 0 hovedansvarlig for fragment 0 og backup for fragment 1, mens node 1 er hovedansvarlig for fragment 1 og backup for fragment 0. Figur 2.3 illustrerer dette. I vanlig drift utføres operasjoner på primærfragmentet, mens en logg over operasjonene også sendes til backup slik at denne kan holdes oppdatert. Figuren viser også at databasen har to redundansenheter. En redundansenhet inneholder en fullstendig kopi av databasen, og i figuren vil node 0,2,4,... og node 1,3,5,... være to redundansenheter, da speilingen gjør at settene inneholder fullstendige kopier av databasen.

Fordelen med en slik arkitektur er at systemet tåler at en node krasjer. Skulle



Figur 2.3: Fordeling av data på noder

node 0 falle bort, er fragment 0 tilgjengelig hos node 1 som da kan overta node 0 sin oppgave. Dette gjør at systemets MTTR blir mye mindre enn den hadde vært dersom data ikke ble tilgjengelig før node 0 hadde omstartet. Men dersom MTTR skal bli liten, er det også viktig at en nodekrasj oppdages hurtig. Ved bruk av *I'm alive*-protokollen [GR93], sender nodene jevnlig meldinger til hverandre. Dersom det ikke mottas meldinger fra en node, besluttes det etter en tidsperiode at noden har feilet, og backup-noden tar over.

Systemet i figur 2.3 er singelfeil-tolerant med tanke på nodefeil. Singelfeil-toleranse vil i dette tilfellet si at systemet tåler en nodefeil uten at hele eller deler av systemet er utilgjengelig. Systemet vil også i mange tilfeller kunne takle flere feil, men dersom to noder som speiler hverandre krasjer samtidig blir noe data utilgjengelig. Når en node har krasjet, er ikke systemet singelfeil-tolerant lengre. For at systemet igjen skal bli singelfeil-tolerant med tanke på nodekrasj, må noden som krasjet komme tilbake i systemet. Dette kan gjøres på to måter. Dersom noden bare var midlertidig ute av drift, vil den kunne komme tilbake i systemet og overta sine gamle oppgaver. Dersom noden ikke kommer opp igjen, kan såkalte *sparenodes* benyttes. En sparenode er en reservenode som ikke brukes i normal drift. Når en node krasjer og ikke kommer opp igjen, kopieres data over til sparenoden som etterhvert overtar som den krasjede noden i systemet. Denne kopieringen krever effektiv kommunikasjon for å få sparenoden hurtig opp i drift.

Legg merke til at arkitekturen over bare garanterer at systemet er singelfeil-tolerant med tanke på nodekrasj. Skal systemet være singelfeil-tolerant, må eksempelvis også nettverk og strømuttak dupliseres. Speilnodene kan benytte forskjellige UPSer og hver node kan har to nettverkskort koblet til to forskjellige

switcher som også har adskilte strømuttak.

2.4 Logg

For at et databasesystem skal komme tilbake i riktig tilstand etter en krasj, logger de fleste databaser operasjoner til disk før transaksjonen får comitte. Dette kalles *write-ahead logging* (WAL). I en parallell database må i tillegg loggen være uavhengig av lokasjon og fragment slik at loggen kan brukes både på primær og backupnode for et fragment. *Location- and Replication Independent Recovery*-strategien (LRI) beskrevet i [Hva99], er en loggmetode som tar hensyn til dette.

De siste 20 årene har CPU-ytelsen økt kraftig, mens diskytelsen ikke har holdt tritt. Dette medfører at disk i mange systemer blir en flaskehals. I den parallelle shared-nothing databasen Sun HADB [HyTBH95] benyttes derfor NWAL eller *Neighbour Write-Ahead Logging*. Loggen sendes over nettverket til en nabonode før commit, noe som er hurtigere enn å skrive til disk.

2.5 Kommunikasjon og prosesser

Databaseprogramvaren på en node i et parallelt databasesystem er svært komplisert og består av mange prosesser. I [HyTBH95] beskrives det blant annet at systemet har fire tjenester: *transaction controller, database kernel, node supervisor* og *update channel*. I vanlig bruk kommuniserer en prosess med både prosesser på egen og andre noder. Eksempelvis vil en *transaction controller* kommunisere med *database kernel* på noder som skal utføre transaksjonen, men den vil også kommunisere med speilnodens *transaction controller* slik at denne kan ta over transaksjonsutførelsen ved feil. Samtidig vil *update channel* sørge for at logg sendes over til speilnode.

Kommunikasjonen som flyter mellom nodene i Sun HADB består i stor grad av mange små meldinger [Mic]. Dette medfører at prosessene hele tiden må stå klar til å motta nye meldinger og eventuelt sende svar på disse. Tabell 2.2 viser tall for innsetting av 1 KB og 100 KB BLOB (*Binary Large Object*) [Mic] i Sun HADB. Resultatene viser at de to tilfellene har relativt forskjellige profiler. Ved små BLOBs sendes det mange små meldinger per sekund. Dette kommer av høy gjennomstrømning og at objektene som settes inn er små. Kontrolldata og BLOB-data utgjør dermed relativt like deler av lasten. Ved store BLOBs sendes færre, men større meldinger per sekund. Her utgjør data en stor del av lasten. Dette kommer av at data både sendes for selve innsettingen og for logg hos speilnode, samtidig som gjennomstrømningen senkes.

BLOB-størrelse	Innsettinger pr. sekund	Meldinger pr. sekund	Meldinger pr. insetting	Bytes pr. innsetting	Bytes pr. melding
1 KB	283,4	4128,7	14,6	3896,8	267,5
100 KB	34,5	1551,8	44,9	132607,4	2952,8

Tabell 2.2: Meldinger og størrelse ved innsetting av BLOB i Sun HADB

2.6 Ytelse

Ved spesifisering av ytelsen til et OLTP-system (OnLine Transaction Processing) er det vanlig å oppgi:

1. Type last
2. Gjennomstrømning (*throughput*)
3. Responstid (*latency*)

For Sun HADB ble det eksempelvis spesifisert at 95% av transaksjonene skulle ha en responstid på under 15 ms for en TPC-B liknende last [HyTBH95].

Ved ytelsesøkning vil enten gjennomstrømning, responstid eller begge endres. Legg merke til at de to størrelsene er avhengige av hverandre. En metode for å sammenligne ytelsen til forskjellige systemer, er derfor å øke lasten på systemet til responstiden når en forhåndsbestemt grense, for så å sammenligne gjennomstrømning.

Ettersom et parallelt databasesystem er avhengig av kommunikasjon for transaksjonsutførelse, og eventuelt også NWAL (se kapittel 2.4), er ytelsen svært avhengig av et effektivt nettverk. For et nettverk er det spesielt tre egenskaper som i stor grad bestemmer ytelsen:

- Transporttid for overføring av en melding
- Overføringsrate (Mbit/s)
- CPU-tid brukt under sending og mottak

Det er innlysende at responstiden for en transaksjon direkte påvirkes av hvor lang tid det tar å sende data over lederen. Høy overføringsrate øker ytelsen fordi flere transaksjoner kan sende meldinger samtidig, noe som medfører høyere gjennomstrømning. I de fleste tilfeller er det en sterk sammenheng mellom transporttid og overføringsrate, men det finnes også eksempler der en kan øke overføringsraten uten å påvirke transporttiden nevneverdig. Et eksempel på dette er økning av antall par ved fiberoverføringer slik at flere pakker kan sendes samtidig.

CPU-tiden brukt ved sending av en melding over nettverket påvirker både transporttid og overføringsrate. Transporttid kan uttrykkes som:

$$\begin{aligned} &t_{cpu_s} = \text{CPU-tid brukt for å sende en melding} \\ &t_{leder} = \text{Transporttid for en pakke over lederen} \\ &t_{cpu_m} = \text{CPU-tid brukt for å motta en melding} \\ &\text{Total transporttid} = \ t_{cpu_s} + t_{leder} + t_{cpu_m} \end{aligned} \tag{2.2}$$

Likning 2.2 viser at transport av meldinger mellom noder påvirkes direkte av hvor mye tid som brukes lokalt på noden ved sending og mottak.

Dersom systemet bruker for mye CPU-tid ved sending av en pakke, vil overføringsraten ikke kunne utnyttes da systemet rett og slett ikke klarer å levere data til kommunikasjonskanalen hurtig nok.

2.7 Oppgaven

Mens overføringsraten mellom prosessor og primærminne de siste årene har steget til over 10 Gbit/s, er 100 Mbit Ethernet fortsatt den vanligste nettverkshastigheten. 100 Mbit/s er tilstrekkelig for standard kontorapplikasjoner, mens parallelle databasesystemer stiller mye høyere krav til overføringsraten.

I Sun HADB viser det seg at systemet ved stor last bruker over 30% av CPU-tiden til meldingssending [Mic]. I en undersøkelse [Hav02a] utført med bruk av 1 Gbit/s Ethernet-kort er det også blitt målt at ved 200 Mbit/s brukes 30% av CPU-tiden til meldingssending.

Tallene i tabell 2.2 viser at båndbredden ikke utnyttes i de to tilfellene. Dersom CPU-tid brukt under sending av meldinger reduseres, vil systemet kunne utnytte denne tiden til andre oppgaver. Dette vil medføre at systemet kan oppnå en høyere gjennomstrømning fordi den ekstra tiden kan brukes til utførelse av transaksjoner. Dersom CPU-tid brukt ved sending og mottak reduseres, vil dette også kunne medføre lavere responstid for transaksjonene (se likning 2.2). Høyere gjennomstrømning vil også medføre høyere antall meldinger som sendes hvert sekund, og systemet vil dermed kreve høyere båndbredde ved bruk av mindre CPU-krevende kommunikasjon.

Det vil i denne oppgaven vurderes om parallelt shared-nothing databasesystem vil oppnå ovenfornevnte effekter ved bruk av InfiniBand.

Kapittel 3

Kommunikasjon

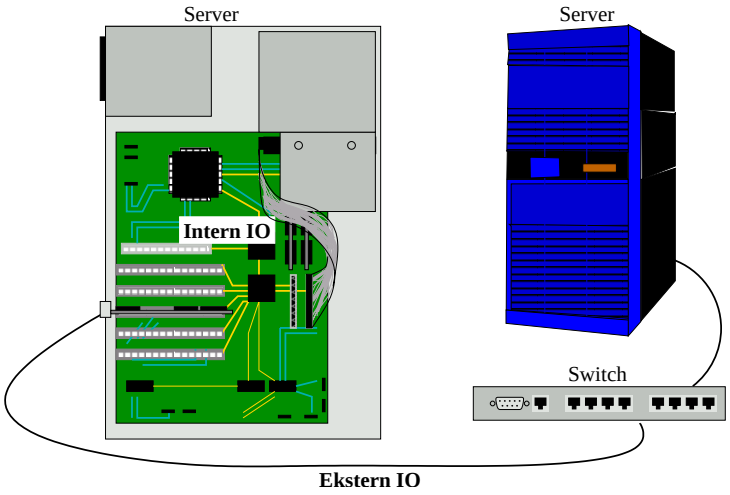
Dette kapitlet beskriver inndeling av kommunikasjon videre i rapporten og busstyper og semantikk for kommunikasjon mellom to enheter.

3.1 Kommunikasjonstyper

Kommunikasjon utført av en datamaskin kan deles inn i to hovedgrupper:

- Intern kommunikasjon
- Ekstern kommunikasjon

Figur 3.1 illustrerer forskjellen mellom de to kommunikasjonstypene.



Figur 3.1: Ekstern og intern IO

Intern kommunikasjon er kommunikasjon mellom enhetene internt i en datamaskin. Det kan for eksempel være overføring av data mellom minne og prosessor eller overføring av data fra CPU til skjermkort.

Ekstern kommunikasjon er kommunikasjon mellom datamaskiner eller andre eksterne enheter. Det kan for eksempel være en vanlig Ethernet nettverkstilkobling mot Internett, modem, firewire mot digitalt videokamera eller fibertilkobling mot et SAN (Storage Area Network).

3.2 Busstyper

En buss er her en kommunikasjonskanal mellom to enheter på et hovedkort. Eldre busser bruker gjerne en "delt buss"-arkitektur. Ved bruk av delt buss kan bare en enhet bruke bussen om gangen. Ved bruk av PCI (kapittel 4.1) benytter enhetene såkalt arbitrering for å fortelle at de ønsker å bruke bussen. Når en enhet har kontroll over bussen, sender den de data den ønsker for så å gi slipp på bussen. Fordelen med denne formen for buss er at den er relativt enkel å implementere for små systemer. I større systemer får derimot denne busstypen problemer med blant annet skalering, fart og fysisk lengde [Mel03].

Ved pakkeswitchet "punkt-til-punkt"-buss benyttes en form for *pipelining* eller samlebåndsteknikk. Hver enhet er koblet til en switch og sender pakker som blant annet inneholder mottakers adresse. Switchen mottar pakker fra enhetene og videresender disse til riktig enhet. Ved bruk av denne teknikken vil ikke en enhet legge beslag på hele bussen ettersom flere segmenter av bussen kan brukes til dataoverføring samtidig. En annen fordel er at forskjellige deler av bussen kan operere på forskjellige hastigheter, men dette krever flytkontroll slik at raske segmenter ikke overlaster tregere segmenter. Bruken av switcher gjør at denne bussformen overkommer de begrensninger i forhold til lengde, fart og skalering som "delt buss"-arkitekturen har.

3.3 Kommunikasjonssemantikk

Kommunikasjon mellom to enheter i et nettverk kan være forbindelsesorientert (*connection-oriented*), eller forbindelsesløs (*connectionless*). Ved forbindelsesorientert kommunikasjon må forbindelsen opprettes før kommunikasjonen kan starte. Dette er analogt med ved vanlig telefoni, hvor brukeren først må slå nummer og vente på svar. TCP/IP er et eksempel på en forbindelsesorientert kommunikasjonsprotokoll.

Ved forbindelsesløs kommunikasjon kreves ingen oppkobling. Pakkene routes gjennom nettverket uavhengig av hverandre. Dette er analogt med å sende

brev gjennom postvesenet. UDP/IP er et eksempel på en forbindelsesløs kommunikasjonsprotokoll.

Det skilles også mellom pålitelig (*reliable*) og upålitelig (*unreliable*) kommunikasjon. Denne inndelingen representerer ytterpunkter for tjenestekvalitet. Ved pålitelig kommunikasjon garanteres det at mottaker får hver pakke som blir sendt. Dette utføres ved bruk av kvitteringspakker (*acknowledgement*) som bekrefter at en pakke er mottatt. Pålitelig kommunikasjon garanterer også at pakkene kommer i riktig rekkefølge og ikke inneholder feil. Garantien for riktig rekkefølge innebærer også at duplikater elimineres. Et duplikat kan f.eks. oppstå dersom en ack-pakke ikke kommer frem og pakken sendes på ny. Ved upålitelig kommunikasjon gis ingen av de ovenfornevnte garantier.

I et høytilgjengelig parallelt shared-nothing databasesystem foretrekkes forbindelsesløs pålitelig kommunikasjon. Ved bruk av denne typen kommunikasjon unngås ekstraarbeid forbundet ved oppsetting og nedkobling av forbindelser. Avhengig av implementasjon, vil også en prosess i databasesystemet slippe å lese fra flere forbindelser, noe som gjør implementasjon enklere. Det kan likevel være ønskelig med en form for pakkeprioritering slik at f.eks. kontrollmeldinger kommer før datameldinger.

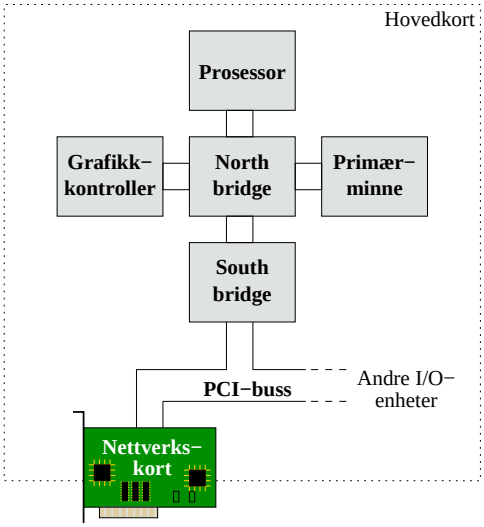
Pålitelig kommunikasjon gir også implementasjonsmessige fordeler. Dersom det kan garanteres at pakker ikke blir endret under transport, kan feilsjekking av meldinger utelates i høyere lag. Det må her nevnes at feilsjekking av data likevel må implementeres for å detektere minnefeil i et høytilgjengelig pålitelig system.

Det er også ønskelig at pakkene ankommer mottaker i samme sekvens som de ble sendt og at pakkene ikke blir duplisert. Sjekk av dette kan i så tilfelle utelates fra høyere lag i systemet.

Kapittel 4

Teknologi for intern kommunikasjon

Intern kommunikasjon, eller intern I/O (input/output), er kommunikasjon mellom enheter internt i maskinen. Figur 4.1 viser en skjematisk skisse av et hovedkort i en vanlig datamaskin. De to viktigste elementene for intern kommunikasjon er broer og busser. En buss er en kommunikasjonskanal mellom enheter, mens en bro er en overgang mellom to eller flere busser. Figuren viser to viktige broer kalt northbridge og southbridge. Northbridge, eller minnebroen, tar seg av kommunikasjon mellom CPU, minne og grafikkontrolleren (skjermkortet). Southbridge, eller IO-broen, tar seg av kommunikasjon med blant annet harddisker, lydkort og nettverkskort via PCI-bussen, og bussen mellom northbridge og southbridge er også en PCI-buss. Northbridge muliggjør også DMA



Figur 4.1: Standard chipset og sti på hovedkort

(*Direct Memory Access*), slik at for eksempel kommunikasjon mellom harddisk og minne kan gjøres uten at data går via CPU.

Nyere servere er i ferd med å gå bort fra oppsettet vist i figur 4.1, da PCI-bussen ikke lenger er rask nok. Videre i dette kapitlet gis først en kort beskrivelse av PCI, og deretter presenteres teknologi som kan erstatte denne internt på hovedkortet.

4.1 PCI

PCI-bussen (*Peripheral Component Interface*) ble lansert av PCI-SIG (PCI Special Interest Group) [Pci] i 1992 som en arvtager for ISA-bussen. Den har siden blitt standard på de fleste hovedkort. PCI-bussen er koblet til southbridge på hovedkortet (se figur 4.1), og standard ekpansjonskort som lydkort og nettverkskort kobles til denne via spor på hovedkortet.

PCI er en delt hierarkisk buss. Rotnoden i hierarkiet kalles vert, mens løvnodene er enhetene koblet til bussen. For å øke antall enheter bussen kan håndtere, brukes *PCI-to-PCI*-broer. Fra rotnivå gis alle enheter samtidig en unik minneadresse slik at systemet får en global oversikt over ressursene. Dette kalles gjerne *memory-mapped I/O*. Denne inndeling fungerer bra for et en-prosessor system der det bare er en vert, men er vanskelige å skalere til systemer med flere verter (dvs. flere CPUer) [Mel03].

Overføring over bussen skjer parallelt, det vil si PCI benytter mange ledere og overfører 1 bit over hver leder samtidig. Parallell overføring stiller store krav til synkronisering, noe som setter begrensninger både for lengde og overføringsrate.

PCI-bussen var i utgangspunktet 32 bit bred og opererte på 33 MHz, noe som gir en overføringsrate på ca. 133 MByte/s. I den senere tid har både bredde og klokkefrekvens blitt økt til henholdsvis 64 bit og 66 MHz, noe som gir en overføringsrate på ca 533 MByte/s (PCI versjon 2.2). Ved 33 MHz tar det 270 ns å overføre data over bussen [Cpq99]. Da 66 MHz versjonen bruker samme protokoll, antas denne å bruke ca 135 ns på samme overføring.

4.2 PCI-X

PCI-X [Cpq99] ble utviklet av Compaq, Hewlett Packard og IBM for å øke ytelsen i kraftige servere. PCI-SIG [Pci] standardiserte spesifikasjonen, og den offisielle PCI-X 1.0 spesifikasjonen ble klar i 1999.

Teknologien er, som navnet tilsier, basert på PCI som den også er bakoverkompatibel mot. PCI-X benytter, på samme måte som PCI versjon 2.2, 64 bit

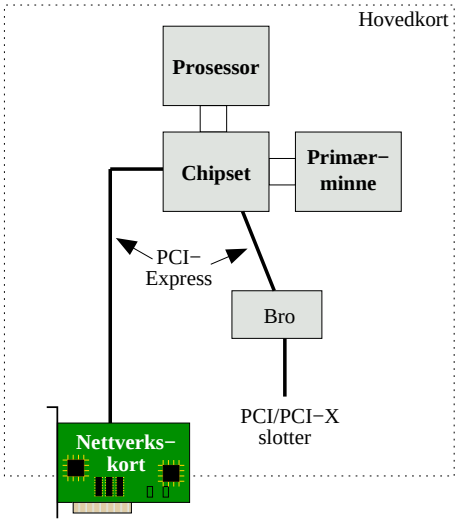
bussbredde, men øker klokkefrekvensen til 133 MHz. Dette gir en overføringshastighet på ca. 1066 MByte/s. Enkelte protokollforbedringer inngår også i standarden. En svakhet ved PCI-X er at hver kontroller bare kan ha én enhet tilkoblet dersom den skal operere på 133 MHz [Mel03]. Ved senkning til 100 MHz hastighet, støttes to enheter, og ved ytterligere senkning til 66 MHz, kan systemet ha opp til 4 enheter pr. kontroller. Ved 133 MHz tar det 75 ns å overføre data over bussen [Cpq99].

I 2002 lanserte PCI SIG spesifikasjonen for PCI-X versjon 2.0 som øker klokkefrekvensen til 266 MHz og 533 MHz. Logisk sett er også denne versjonen kompatibel med PCI, men den opererer bare på 1,5 og 3 volt og er dermed ikke kompatibel med 5 volts PCI-enheter. En spesifikasjon for en 1066 MHz versjon av PCI-X forventes lansert i løpet av 2004, og en 2133 MHz versjon er under vurdering av PCI-SIG [PSi].

4.3 PCI-Express

PCI-Express, eller 3GIO (3rd Generation IO), ble utviklet av Arapahoe Working Group. Spesifikasjonen ble utgitt av PCI-SIG juli 2002 [Pci].

Hensikten med PCI-Express er å tilby et alternativ til PCI/PCI-X som er raskere samtidig som det reduserer antall ledere, noe som blant annet gir lavere produksjonskostnad [Mel03]. PCI-Express er software-kompatibel med PCI/PCI-X slik at operativsystemer skal kunne kjøres uendret. PCI SIG presiserer at PCI-Express ikke skal erstatte PCI/PCI-X. Det ser derimot ut til at PCI-Express vil erstatte AGP-porten på hovedkortene ettersom Intels nyeste



Figur 4.2: Systemoppsett ved bruk av PCI-Express (basert på figur fra [Bha02])

brikkesett har utelatt denne [no-04].

PCI-Express-arkitekturen kan deles inn i tre lag:

1. Fysisk lag
2. Data-link lag
3. Transaksjonelt lag

Disse lagene beskrives nærmere i de neste delkapitlene.

4.3.1 Fysisk lag

Den kanskje største forskjellen mellom PCI/PCI-X og PCI-Express er at PCI-Express benytter seriell overføring i motsetning til parallell overføring som benyttes av PCI/PCI-X. Den serielle overføringen skjer over to par ledere, ett par i hver retning, og hastigheten per par er 2,5 Gbit/s (2,0 Gbit/s effektivt [PSi]). Dette gir en hastighet på 125 MByte/s per leder i motsetning til 11,5 MByte/s per leder for PCI-X. Den serielle overføringen er pakkebasert og signalet over kablene har integrert klokkesignal og benytter 8b/10b koding [EM99]. Det integrerte klokkesignalet gjør at PCI-Express slipper å klokke bussegmenter etter tregeste enhet slik som tilfellet er for PCI/PCI-X.

For å oppnå høyere hastighet kan bussen utvides med flere par i hver retning. Ett par i hver retning kalles 1X, og antall par kan dobles opp til 32X. Ved bruk av flere par, fordeles data byte-vis på parene etter tur.

PCI-Express går bort i fra det chipsettet som er vist i figur 4.1. I oppsettet vist i figur 4.2 spares et busshopp ved at nettverkskortet kobles direkte til brikken som sørger for kommunikasjon mot minnet, noe som anslås å senke intern transporttid med over 40% [Bha02].

PCI-Express opererer også på lavere voltstyrke enn sine forgjengere. Dette er blant annet heldig med tanke på kjøling av systemet.

4.3.2 Data-link-lag

Data-link-laget sørger for pålitelig leveranse av pakker over bussen. En flytkontroll-protokoll brukes for å påse at mottaker har ledige buffere, noe som medfører at pakker ikke sendes forgjeves over bussen. Data-link laget benytter også sekvensnumre og CRC for å sikre at pakker kommer i riktig rekkefølge og ikke er korrupte. Dersom en pakke er korrupt, blir den sendt på ny, noe som skjules for høyere lag.

4.3.3 Transaksjonelt lag

Det transaksjonelle laget håndterer forespørsler fra høyere lag. Pakkeheaderen, som legges til i dette laget, inneholder blant annet en 32 eller 64 bit adresse og en unik identifikator slik at pakker fra data-link laget kan sendes til riktig mottaker i høyere lag.

Laget benytter splitta transaksjoner, dvs. en overføring deles i to transaksjoner: en transaksjon for utsending av forespørsel og en for svaret. Videre kan pakkene tilordnes prioritet, samt *no-snoop* og *relaxed-ordering* (se [Gru03]).

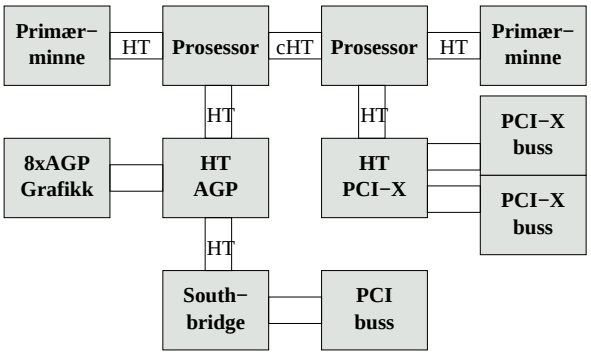
4.4 HyperTransport

AMD startet utviklingen av HyperTransport [Amd01] i 1997, og i 2001 ble HyperTransport Consortium stiftet av blant annet AMD, Apple Computer, Cisco Systems, NVIDIA og Sun Microsystems. Blant målene for designet var høyere intern båndbredde for I/O, økt skalerbarhet, minimale softwarekrav og mindre strømforbruk. HyperTransport er utviklet for å være software-kompatibel med PCI.

En egen variant av HyperTransport er utviklet for bruk mellom prosessorer (se figur 4.3). Denne kalles *coherent HyperTransport* (cHT) og skal sørge for koherens i cache til prosessorer i et multiprosessor system.

4.4.1 Fysisk lag

På samme måte som PCI-Express, går HyperTransport bort fra delt buss. HyperTransport benytter seg av to sett ledere for parallell sending og mottak av data (ett sett i hver retning). Den benytter egne ledere for klokkesignal og



Figur 4.3: Systemoppsett ved bruk av HyperTransport (basert på figur fra [Kra02])

kontrollsignaler, noe som medfører at den bruker langt flere ledere enn PCI-Express. Eksempelvis gir to par i hver retning 24 ledere til sammen [hyp01]. Bussbredden kan varieres fra 2 til 32 bit. Overføringsraten for to par er ca. 800 Mbit/s og ca. 89,6 Gbit/s ved 32 bit bredde. Pakkestørrelsen er multiplum av 4 byte. HyperTransport overfører data både på stigende og synkende klokkeflanke.

4.4.2 Data-link lag

Data-link-laget tar seg av initialisering, konfigurasjon og flytkontroll. Det foretas også feildetektering ved bruk av CRC.

4.4.3 Protokoll/transportlag

På transportnivå bruker HyperTransport datastrømmer (*streams*), og bussen kan håndtere flere strømmer samtidig. Noder på bussen kan lenkes og det er derfor opp til en enhet å sjekke om informasjon den mottar på bussen er ment for den selv eller skal videresendes. Pakkene identifiseres til strømmer ved bruk av en identifikator, og det kan være opptil 32 forskjellige identifikatorer på bussen samtidig.

HyperTransport benytter 40 bits adressering, og datadelen av en pakke har mellom fire og 64 byte størrelse.

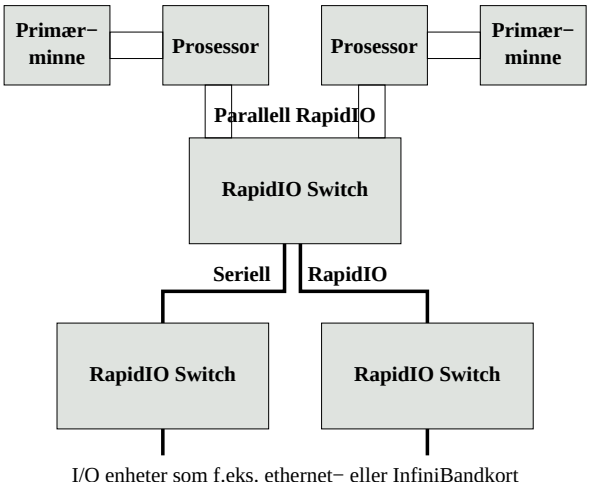
4.5 RapidIO

RapidIO [Bou03] ble originalt utviklet av Motorola og Mercury Computer Systems, og i år 2000 ble RapidIO Trade Association stiftet. I etterkant er blant annet IBM og Ericsson blitt medlem av organisasjonen.

RapidIO har som mål å bli den mest populære tilkoblingsbussen for enheter internt i serveren. Den benytter memory mapped I/O (se kapittel 4.1) slik at operativsystemer skal kunne brukes uten særlig endring. Antall ledere reduseres sammenlignet med PCI, og enkle bitfeil, samt flere typer flerbitfeil, maskeres.

RapidIO er pakkeswitchet og finnes i en parallell og en seriell variant. Den serielle varianten finnes i utgaver med 1 bit og 4 bit bredde, noe som gir hastigheter opp mot henholdsvis 5 Gbit/s og 20 Gbit/s. Den parallelle varianten opererer i 8 bit eller 16 bit modus, og kan oppnå henholdsvis 32 Gbit/s og 64 Gbit/s. Eksempler på bruken av parallell og seriell RapidIO er vist i figur 4.4.

Datadelen av pakkene kan være fra 1 til 256 byte i størrelse, og adressestørrelsen er 8 eller 16 byte. Pakkene inneholder også en 16 byte CRC for detektering



Figur 4.4: Eksempler på bruk av RapidIO (basert på figur fra [Bou03])

av pakkefeil. På protokollnivå har RapidIO en rekke forskjellige transaksjonstyper, deriblant transaksjoner for cache-koherens. Transaksjoner kan gis forskjellige prioritet.

RapidIO har innebygd meldingsstøtte, det vil si at meldingsmottak og utsending er implementert i hardware (men synlig for software). Dersom systemet har flere prosessorer med eget minne koblet til RapidIO-bussen, kan meldinger brukes for å utveksle data mellom prosessorene.

Flytkontroll er implementert i hardware. Flytkontrollen inneholder blant annet mekanismer for resending av korrupte meldinger og bremsing av avsender dersom mottaker ikke kan motta pakkene hurtig nok.

Kapittel 5

Teknologi for ekstern kommunikasjon

Ekstern kommunikasjon er kommunikasjon som foregår mellom datamaskiner. Dette kapitlet beskriver utvalgt teknologi for ekstern kommunikasjon som er tilgjengelig i dag.

5.1 Ethernet

Ethernet ble utviklet av Xerox Corporation på 1970-tallet og baserer seg på Carrier Sense Multiple Access Collision Detect (CSMA/CD) protokollen. Digital Equipment Group, Intel Corporation og Xerox Corporation utviklet rundt 1980 spesifikasjonen for 10 Mbit/s Ethernet. IEEE 802.3 [IEE] standarden baserte seg på denne og ble i 1985 en offisiell standard. I dag anslås det at ca. 85% av alle lokalnettverk består av Ethernet-maskinvare [cis04].

IEEE 802.3 standarden definerer tre hastigheter:

- 10 Mbit/s - 10Base Ethernet
- 100 Mbit/s - Fast Ethernet
- 1000 Mbit/s - Gigabit Ethernet

I tillegg definerer IEEE 802.3ae en ny 10 Gigabit standard.

Ethernet gjøres tilgjengelig ved at et nettverkskort kobles til PCI-bussen. Nyere hovedkort kan også ha Ethernet-maskinvare direkte integrert i hovedkortet.

5.1.1 Media Access Control

IEEE802.3 deler Ethernet i tre lag:

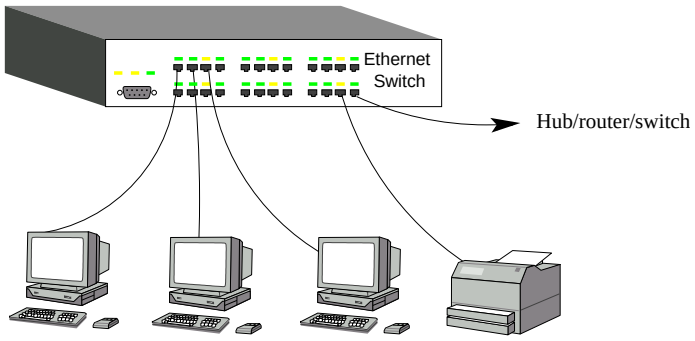
- Fysisk lag
- MAC (Media Access Control)
- MAC-klient

Det fysiske laget oversetter og sender frames (pakker) over transmisjonsmediet. Ved bruk av vanlig *twisted pair*-kabler, sendes signalet som en rekke strømpulser, mens det ved bruk av fiber må oversettes til lyssignaler. *Twisted Pair*-kabel er det vanligste transmisjonsmediet for Ethernet. Navnet gjenspeiler at en vanlig kategori 5 *Twisted Pair*-kabel består av 4 par, der de to lederene i et par er tvinnet om hverandre, samtidig som alle parene også er tvunnet om hverandre. Fast Ethernet benytter to par i en kategori 5 kabel. Ethernet kan også sendes over optisk kabel.

MAC-laget har ansvar for innkapsling av data i frames, det vil si det skal legge på riktige headere før utsending. Headerne inneholder blant annet lengden på datadelen av i pakken, kildeadresse og mottakeradresse. Maksimal pakkestørrelse er 1500 byte. Adressene kalles MAC-adresser og er unike 48 bits hardkodede adresser tildelt ved produksjon. De 3 første bytene av MAC-adressene identifiserer produsent av maskinvaren, mens de 3 gjenværende er maskinvarens serienummer. Ved mottak er MAC-laget ansvarlig for parsing og feildetektering av en frame. MAC-laget implementerer også CSMA/CD protokollen. Oppsummert fungerer protokollen på følgende måte:

- **Carrier Sense (CS) -** Nettverksenhetene skal lytte på nettverket for å oppdage når det er ledig
- **Multiple Access (MA) -** En hvilken som helst enhet kan begynne å sende data når nettverket er ledig
- **Collision Detect (CD) -** Hvis to enheter begynner å sende en frame samtidig, vil de ødelegge for hverandre. Det oppstår en kollisjon. Alle enhetene i nettverket må derfor kunne detektere kollisjoner. Ved kollisjon skal enhetene avbryte sendingen umiddelbart og vente en liten tilfeldig tidsperiode før de prøver på ny.

IEEE802.3 krever i utgangspunktet bare at en nettverksenhet skal støtte halv duplex, men nyere nettverkskort har også støtte for *full duplex*. Ved full duplex kan enhetene sende og motta data samtidig, og begge enhetene i endene av en kabel kan sende samtidig uten at det oppstår kollisjon. Ved overføring over



Figur 5.1: Switch i et Ethernet-lokalnettverk

twisted pair-kabel utføres dette ved at to ledere brukes til mottak (RX) og to til sending (TX). Ved bruk av full duplex blir dermed CSMA/CD-protokollen overflødig.

MAC-klient-laget tilbyr grensesnitt mot MAC for høyere lag i protokollstakken. For nettverkskort kalles laget Logical Link Control, og det er beskrevet i standard IEEE 802.2.

5.1.2 Switching

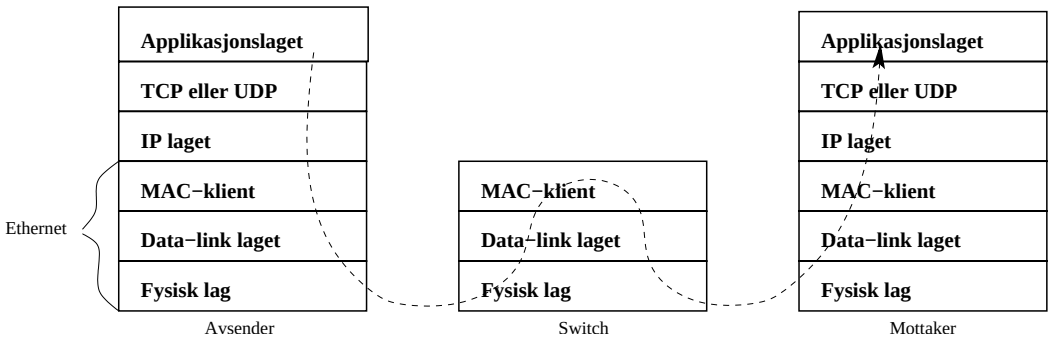
Huber, switcher og routere utgjør knutepunktene i et Ethernet-nettverk. Nettverksklienter (datamaskiner, printere og lignende) kobles til en switch som igjen kan kobles til switcher tilknyttet andre nettverksklienter. Figur 5.1 illustrerer dette.

Mens en hub bare videresender pakker den mottar til alle tilknyttede enheter, er en switch langt mer avansert. En switch vedlikeholder en liste over hvor bestemte MAC-adresser befinner seg og videresender bare pakker på riktig port (kontakt/tilkoblingspunkt). Unntaket er dersom den ennå ikke har lært hvor en bestemt MAC-adresse befinner seg. I de tilfellene sender den pakken videre på alle porter. I tillegg kan mange switcher bufre pakker og på den måten unngå enkelte pakkekollisjoner.

En router knytter sammen flere lokalnettverk. Mens switchen opererer på MAC-nivå, vil en switch også operere på høyere nivå (IP-laget).

5.1.3 IP - Internet Protocol

IP er den protokollen som hovedsaklig brukes sammen med Ethernet i dag, men det finnes også implementasjoner av denne protokollen for bruk mot andre nettverksteknologier. Protokollen ble utviklet for bruk i større nettverk, noe navnet også tilsier. Da denne oppgaven dreier seg om kommunikasjon mellom



Figur 5.2: Lagdeling og pakkesti ved sending og bruk av IP

noder i en parallell database, er det nærliggende å anta at disse nodene befinner seg på samme lokalnettverk. Det vil derfor bare bli gitt en beskrivelse av IP for mindre nettverk. For en fullverdig beskrivelse av IP, se [Tan96].

Figur 5.2 viser lagdelingen i et nettverk som benytter IP. Enheter i et IP-nettverk tildeles unike adresser kalt IP-adresser. Adressestørrelsen for IP versjon 4, som er den vanligste varianten i bruk i dag, er på 32 bit. Ved sending av en pakke, må IP-laget oversette IP-adresser til MAC-adresser slik at Ethernet-lagene kan route pakken riktig gjennom nettverket (switcher opererer som nevnt bare på MAC-nivå). Dette skjer via ARP-protokollen. Kort forklart kringkaster enheten som skal sende en pakke, en forespørsel om hvilken enhet som har IP-adressen den skal sende til. Enheten som har denne IP-adressen svarer på forespørselen og pakken som skulle sendes kan nå routes til riktig sted. Resultater av ARP-forespørsler caches i en periode for bruk ved videre kommunikasjon.

IP-laget legger på IP-headere før pakken overleveres til MAC-klient-laget. Disse headerne inneholder blant annet total pakkelengde, avsender og mottakers IP-adresser og en checksum for detektering av feil i headeren. En IP-pakke har en maksimalstørrelse på 64 Kbyte.

Til overliggende lag tilbyr IP en upålitelig pakketjenteste. Det gis ingen garanti for at pakkene kommer frem til mottaker, kommer frem i riktig rekkefølge eller har datainnhold som ikke er korrupt.

TCP - Transmission Control Protocol
TCP [Tan96] tilbyr pålitelig forbindelsesorientert kommunikasjon over IP. Den garanterer at pakker kommer frem (dersom det er mulig), og sørger for at pakker kommer i riktig sekvens og med korrekt meldingsinnhold.

UDP - User Datagram Protocol
UDP [Tan96] tilbyr upålitelig forbindelsesløs kommunikasjon over IP. Upålitelig betyr i denne sammenheng at den verken garanterer at pakker kommer frem i riktig sekvens eller i det hele tatt kommer frem. Den har likevel mulighet

for å sjekke at de meldinger som kommer frem ikke er korrupte.

Ettersom Internett har vokst, har adresserommet brukt i IP versjon 4 blitt for lite. En ny versjon kalt IP versjon 6, eller IPv6, er derfor under innføring. Denne benytter 128 bit adresser og har blant annet utbedret støtte for Quality of Services.

5.2 10 Gigabit Ethernet

10GbE (10 Gigabit Ethernet) er spesifisert i IEEE 802.3ae standarden. Den støtter bare full duplex og overføring via fiber, noe som overflødiggjør CSMA/CD. Sett bort i fra dette er teknologien ment å være bakoverkompatibel med de tidligere variantene.

Som navnet tilsier, tilbyr denne varianten en overføringshastighet på 10 Gbit/s. Adapterkort for 10GbE finnes for 133 Mhz 64-bit varianten av PCI-X (se kapittel 4.2) selv om bussteknologien ikke støtter større hastigheter større enn 1066 GByte/s.

5.3 Advanced Switching over PCI-Express

Advanced Switching over PCI-Express [Asp03], heretter forkortet AS, er en utvidelse av PCI-Express. Den benytter samme data-link og fysisk lag som PCI-Express, men definerer et alternativt transaksjonslag. Mens PCI, og dermed i stor grad også PCI-Express, har en hierarkisk struktur med prosessoren som rotnode, ønsker AS å tilby en mer peer-to-peer-rettet struktur hvor enhetene kan kommunisere direkte med hverandre. AS tilbyr en struktur som dermed bedre kan støtte opp om for eksempel multiprosessor-systemer (både innad i flerprosessormaskiner og i systemer bestående av mange noder).

AS benytter en egen pakkeheader. Headeren er kompatibel med PCI-Express og inneholder blant annet sekvensnummer, CRC-kode og informasjon for routing gjennom nettet. Routinginformasjonen inneholder ikke avsender eller mottakeradresse, men informasjon til switchene om hvordan de skal videreformidle pakken. I tillegg inneholder headeren en identifikator som sier hva slags protokoll pakkens datainnhold benytter. Pakker i andre protokollformater kan dermed bruke AS som tunnel over PCI-Express.

Prosessor-til-prosessor kommunikasjon kan utføres via SLS-protokollen (*Simple Load Store*). SLS benytter en type adressering mer tilpasset denne formen for kommunikasjon enn det vanlig PCI-Express tilbyr.

AS garanterer at pakker er feilfrie (vha. CRC og retry), samtidig som routeteknikken gjør at pakken kommer frem i samme rekkefølge som de ble sendt.

Da AS benytter samme fysiske lag som PCI-Express tilbyr den en effektiv hastighet på 2 Gbit/s per leder, og i [MR] anslås det en responstid på under 1 mikrosekund for oversending av pakker mellom noder på lokalnettverk og mindre enn 100 meters avstand.

Til høyere lag tilbyr AS to modeller kalt Simple Queueing (SQ) og Secure Data Transport (SDT). SQ tilbyr enkel meldingsutveksling og er sammenlignbar med UDP i at flere enheter kan sende meldinger til samme kø hos en enhet. I motsetning til UDP, er derimot SQ pålitelig. Den garanterer at meldinger kommer i riktig rekkefølge, ikke er korrupte og støtter dataoverføring direkte til applikasjoners minneområde (OS-bypass). UDP kan teoretisk sett implementeres over AS og programmer vil dermed kunne kjøres over AS uten endring. SDT er også pålitelig og implementerer socket-operasjoner direkte i hardware, og TCP kan implementeres over denne.

5.4 SCI

Scalable Coherent Interface [dol96] er definert i ANSI/IEEE Std 1596-1992. Arbeidet med SCI ble påbegynt i 1988. Målet med SCI var å utvikle en buss som hadde høy overføringsrate, lav responstid og hadde lav CPU-bruk samtidig som den var skalerbar. Den skulle også ha støtte for minnekoherens.

SCI benytter enten seriell overføring over fiber eller parallell overføring over kobber, og benytter et ledersett hver vei for å oppnå full duplex. Kortene benytter PCI (32 eller 64 bit, 33 eller 66 Mhz) og har en overføringsrate på 667 MByte/s pr link [dol04]. For applikasjon til applikasjon oppgis en overføringsrate på 326 MByte/s og en responstid på 1.4 mikrosekunder.

Pakkene har en headerstørrelse på 16-byte og datadelen kan ha en størrelse på 16, 64 eller 256 byte. Pakkene har også en 16 bit CRC-kode for deteksjon av korrupte meldinger. Adressene er på 64 bit, der 48 bit brukes til adressering internt på noden. De 16 mest signifikante bitene brukes for å bestemme hvilken node pakken skal til. Dette gir en maksimalstørrelse på 64000 noder for systemet.

Protokollene støtter to former for kommunikasjon:

- Shared-memory
- Meldingsutveksling

Ved bruk av shared-memory, er bruk av SCI transparent i forhold til programvaren på nodene. Nodene har et delt fysisk minneområde som er distribuert på nodene i systemet. Forespørsler til data i dette minneområdet oversettes og

utføres automatisk av SCI-hardwaren. Protokollen har også innebygde funksjoner for cache-koherens. Denne formen for kommunikasjon er ment for systemer av typen shared-memory nevnt under kapittel 2.1.

En ulempe med shared-memory-varianten er at den i høy grad involverer CPU. Meldingsutvekslingsvarianten benytter derimot en innebygd DMA-kontroller for å gjøre overføringene mindre CPU-krevende. DMA-kontrolleren kan overføre data fra minnet på en node til minnet på en annen node uten midlertidig bufring. For å benytte denne formen for kommunikasjon må noe minne reserveres. Dette for å hindre at SCI-adapteren aksesserer minne den ikke har tillatelse til å bruke. Applikasjoner lager en kø over forespørsler som oversendes SCI-adapteren, som så tar seg av selve overføringen via DMA.

Noder kan kobles sammen i ring eller ved bruk av switcher, og responstiden over nettverket er avhengig av valgt topologi. SCI har støtte for redundante linker for høyere tilgjengelighet

SCI har tidligere blitt forsøkt innført i Sun HADB. Dette er beskrevet i [MS00].

5.5 Myrinet

Myrinet [myr04a] er spesifisert i ANSI/VITA 26-1998, og er en teknologi utviklet for bruk i clustrede systemer. Målet med teknologien var å utvikle en løsning som hadde høyere ytelse og tilgjengelighet enn vanlig Ethernet.

Teknologien var originalt basert på 9-bit parallell overføring med ett ledersett i hver retning. I 2000 kom en seriell versjon som benytter 8b/10b koding. Hastigheten for denne versjonen er 2 Gbit/s per retning, og de to ledersettene muliggjør full duplex. Mellom prosesser er det målt en responstid på 4-6 mikrosekunder for små meldinger.

Det er ingen fast definert pakkestørrelse, men maksimal størrelse vil begrenses av for eksempel størrelse på ledig buffer. Samtidig sier spesifikasjonen at nettverket skal ha en maksimal pakkestørrelse (*Maximum Transfer Unit*) på minst 4 MB. Stor pakkestørrelse muliggjør implementasjon av andre protokoller eller transport av andre pakketyper (for eksempel IP) over Myrinet.

Pakkeheaderen inneholder pakketype og routinginformasjon, mens den siste byten i en pakke utgjør CRC-kode for deteksjon av korrupte meldinger. For systemer med flere enn to noder, brukes switcher for å koble sammen nodene. I fysisk lag har Myrinet flytkontroll og en form for vranglås-deteksjon. På samme måte som Ethernet har Myrinet 48-bit MAC-adresser. Myrinet benytter et “heartbeat”-signal for å oppdage feilende nettverksenheter og systemet har støtte for bruk av alternative stier ved feilende switcher eller noder.

Myrinet-spesifikasjonen definerer ikke et meldingsgrensesnitt for bruk mot høy-

ere lag, men Myricom selger en meldingsgrensesnitt kalt GM [myr04b]. GM støtter OS-bypass og tilbyr blant annet pålitelig leveranse av pakker i riktig rekkefølge.

Av større systemer som i dag benytter Myrinet, har National Center for Supercomputing Applications (NCSA) et Myrinet-cluster med 2500 prosessor og Los Alamos National Laboratory har et med 2816 prosessorer [top03].

5.6 QsNet

QsNet [qua04] er utviklet av Quadrics og er laget for kommunikasjon mellom prosessorer i superdatamaskiner. Den nyeste varianten heter QsNet^{II} og benytter PCI-X grensesnitt.

Systemet bruker parallell overføring over fiber, og maksimal lengde på lederne er 100 meter. For mindre distanser kan også kobber benyttes. Maksimal overføringshastighet er 1,3 GByte/s (906 MByte/s etter fjerning av kontrollinformasjon). Det finnes ett ledersett i hver retning slik at systemet kan benytte full duplex.

Adresseområdet er på 64 bit og QsNet^{II} har støtte for Remote Direct Memory Access (RDMA) fra brukerapplikasjon til brukerapplikasjon i 64 bit arkitekturer. Den benytter DMA for overføring mellom adapterkort og minne, og protokollhåndtering utføres i hardware på adapterkortet. Protokollen som benyttes er pålitelig, og kan detektere feil og bruke alternative stier rundt feilende switcher.

På softwaresiden har Quadrics implementert en egen versjon av MPI 1.2 (*Message Passing Interface* - se [SOHL$^+$96]) kalt Quadrics MPI. Operativsystemet kan benytte et meldingslag som blant annet tilbyr IP.

Nettverket bruker switcher og bruker en topologi kalt *radix 4 fat tree* (se [qua04]). Denne topologien ble blant annet valgt da den har mange alternative stier.

Av større systemer som i dag benytter Quadrics, har Pacific Northwest National Laboratory et Quadrics-cluster med 1936 prosessorer, mens Los Alamos National Laboratory har et med 8192 prosessorer [top03].

5.7 Fibre Channel

Utviklingen av Fibre Channel [FIB] ble startet i 1988, og standarden ble godkjent av ANSI i 1994. Fibre Channels utvikling er i stor grad fokusert rundt ekstern lagring i form av SAN (*Storage Area Network*).

Som transmisjonsmedium benyttes kobberbasert eller fiberoptisk kabel, og ved sistnevnte er maksimal avstand 10 km. Nyere Fibre Channel maskinvare tilbyr 4 Gbit/s overføringshastighet og støtter full duplex. Fibre Channel benytter seriell overføring.

Adresserommet er på 24 bit, og på transportnivå benyttes kreditt-basert eller ende-til-ende-flytkontroll avhengig av type transport. Nettverkstopologi kan være direkte link mellom to enheter, flere enheter koblet i ringstruktur eller ved bruk av switcher slik som ved for eksempel Ethernet.

Til operativsystem og applikasjoner tilbys protokoller for overføring av lagringsrelaterte operasjoner som for eksempel SCSI-kommandoer. For node-til-node-kommunikasjon finnes i tillegg FC-VI og IP over InfiniBand. FC-VI er en implementasjon av VIA, eller *Virtual Interface Architecture* [Via97], som er en arkitektur utviklet av Intel, Microsoft og Compaq for meldingsutveksling i et cluster. VIA er designet for å redusere CPU-bruk ved å la applikasjonene i clusteret kommunisere uten at meldingene går om operativsystemet (OS-bypass).

Kapittel 6

InfiniBand

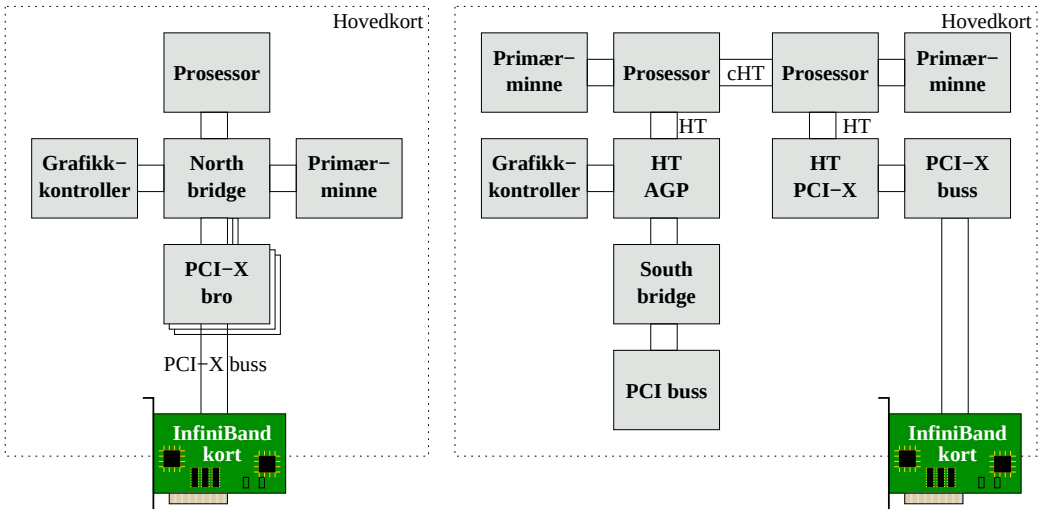
InfiniBand-spesifikasjonen er utformet av InfiniBand Trade Association (IBTA) [IBt]. Denne organisasjonen oppstod da The Future I/O Initiative, bestående av blant annet Compaq, IBM og Hewlett Packard, og The Next Generation I/O Initiative, bestående av blant annet Dell, Hitachi, Intel, Nec, Siemens og Sun Microsystems, bestemte seg for å forene krefter. I august 1999 dannet Compaq, Dell, Hewlett Packard, IBM, Intel, Microsoft og Sun Microsystems The InfiniBand Trade Association der planen var å kombinere de beste ideene fra de to tidligere initiativene. I dag er over 200 forskjellige firmaer verden over medlem av IBTA.

InfiniBand ble opprinnelig utviklet som en erstatter for PCI, det vil si at den kunne brukes både for kommunikasjon mellom enheter internt på hovedkortet og ekstern kommunikasjon. Etterhvert har fokus skiftet til bare ekstern kommunikasjon. En av grunnene til dette kan være at PCI-SIG [Pci] valgte PCI-Express som arvtager/alternativ til PCI og PCI-X.

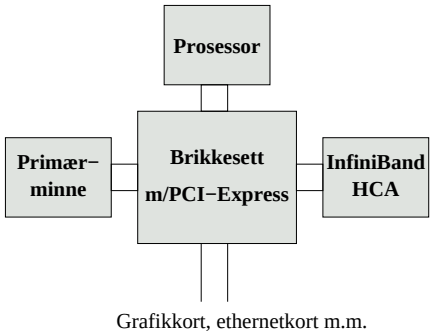
6.1 Topologi

Et InfiniBand-nettverk kan ha opptil fem forskjellige typer enheter:

- Host Channel Adapter (HCA)
- Target Channel Adapter (TCA)
- Switch
- Router
- Subnet Manager



Figur 6.1: InfiniBand-kort tilknyttet et hovedkort via PCI-X



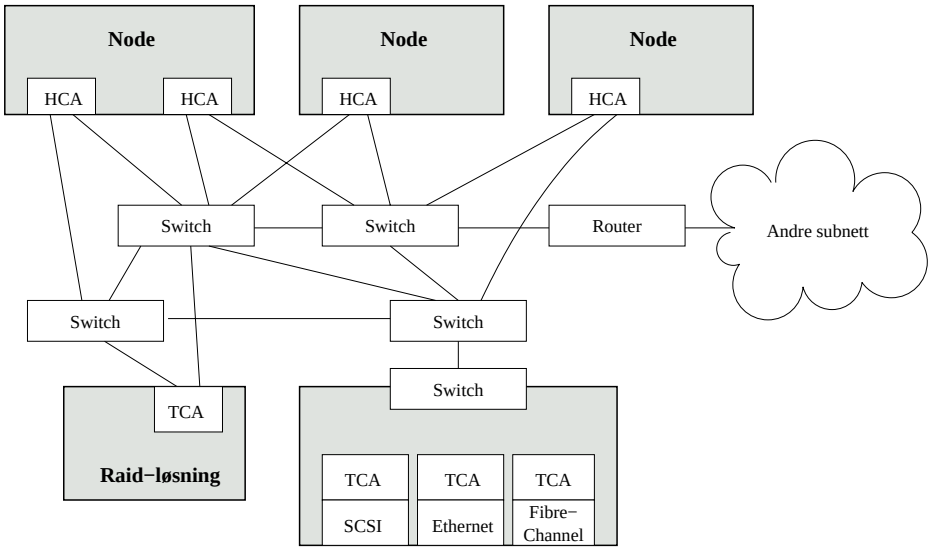
Figur 6.2: InfiniBand-HCA direkte integrert i hovedkort

En HCA er en adapter som befinner seg i serveren og utgjør serverens grensesnitt mot nettverket. Den tilsvarer med andre ord et nettverkskort ved bruk av Ethernet.

En HCA kommuniserer med prosessor og minne internt på serveren via hovedkortets interne buss. I dag tilbys en HCA i form av et PCI-X-kort og produseres blant annet av Mellanox Technologies. Figur 6.1 viser en skjematisk skisse over hvordan en PCI-X HCA kan tilknyttes de andre komponentene i systemet. Senere kan også HCA integreres direkte i hovedkortet og skissen blir da seendes ut som vist i figur 6.2.

En HCA kan være direkte tilknyttet en annen HCA eller TCA, eller den kan være tilknyttet en switch.

En TCA muliggjør tilknytning av I/O-enheter til nettverket. Den kan for eksempel tilhøre en disk- eller tapekontroller, eller den kan være et grensesnitt mot en SAN løsning. En TCA inneholder en I/O-kontroller som er spesifikk



Figur 6.3: InfiniBand nettverk

for enhetens protokoll. En TCA kan tilknyttes en HCA direkte eller den kan kobles til en switch.

En switch er, som beskrevet i kapittel 5.1.2, en enhet som utgjør et knutepunkt i nettverket. En switch er tilkoblet HCAer og TCAer og eventuelt også andre switcher eller routere. Switchene tar seg av routing av pakker i nettverket ved inspeksjon av pakkenes headere og videresending på riktig port.

En router utgjør en overgang mellom to nettverk. Dersom en mottaker ikke finnes på lokalnettet, sendes pakken til routeren. Routeren inspiserer pakken og sender den videre med riktig mottaker-adresse på neste nettverk.

En subnet-manager er ikke en fysisk enhet, men programvare som tar seg av konfigurasjon av nettverket. InfiniBand gjør bruk av høyprioritets Subnet Managment Packets (SMP), og kan ved hjelp av disse:

- Detektere noder koblet til nettet
- Detektere nettverkets topologi
- Sette opp switchenes routingtabeller
- Konfigurere noder og switcher

Dette innebærer også at subnet-manageren detekterer feilende ledere, switcher eller HCAer og bruk av alternative router rundt disse. Subnet-manageren implementeres ofte direkte i switchene. Dersom to HCAer eller en HCA og en TCA er direkte koblet sammen kan subnet-manageren også kjøre på en HCA.

Figur 6.3 viser en mulig topologi i et InfiniBand-nettverk. Legg merke til at en InfiniBand-adapter kan ha to porter og dermed kan kobles til to forskjellige switcher. I et høytilgjengelig system har dette en stor fordel. Dersom en switch eller en ledning skulle bli ødelagt, vil subnet manageren automatisk kunne route pakkene over på den andre porten og dermed omgå de ødelagte enhetene.

Verdens tredje største cluster består av 2200 InfiniBand-baserte noder og eies av Virginia Tech [top03].

6.2 Arkitektur

6.2.1 Fysisk lag

InfiniBand benytter seriell punkt-til-punkt overføring. En 1X InfiniBand-link består av fire ledere (to hver vei), og InfiniBand støtter en overføringsrate på 2.5 Gbit/s i hver retning. Bruken av ett lederpar hver vei gjør at InfiniBand støtter full duplex. Overføringen over en link benytter 8b/10b-koding (se [EM99]), og ca 20% av overført last er da kontrolldata. Effektiv overføringsrate blir derfor ca. 2 Gbit/s. I tillegg til 1X linker, definerer spesifikasjonen også 4X, 12X og 32X linker. Disse har tilsvarende henholdsvis fire, tolv og tredve ganger så mange ledere og dermed tilsvarende økning i overføringsrate. Overføringsratene er oppsummert i tabell 6.1. I tillegg har InfiniBand-kort ofte to porter noe som gjør at overføringsraten kan dobles enda en gang. Ende-til-ende-responstiden er 5,4 mikrosekunder [Sun03]. Til sammenligning har 100 Mbit/s Ethernet en responstid på 60 mikrosekunder.

Lederene kan være laget av optisk fiber eller kobber. Optisk fiber har lengre rekkevidde uten signaltap. Tabell 6.2 viser maksimal kabellengde ved forskjellige hastigheter og material.

Link	Ledere	Overføringsrate	Effektiv overføringsrate	Full duplex overføringsrate
1X	4	2.5 Gbit/s	2 Gbit/s	4 Gbit/s
4X	16	10 Gbit/s	8 Gbit/s	16 Gbit/s
12X	48	30 Gbit/s	24 Gbit/s	48 Gbit/s
32X	128	80 Gbit/s	64 Gbit/s	128 Gbit/s

Tabell 6.1: Overføringsrater over InfiniBand

6.2.2 Link-lag

Link-laget definerer pakkelayout og er ansvarlig for pakkeswitching internt i et subnet. Det finnes to typer pakker: managment-pakker (SMP) og datapakker. Datapakker kan ha datastørrelse opptil 4 Kbyte.

Link	Kobber	Optisk fiber
1X	17 m	10 km
4X	10 m	125 m
12X	10 m	125 m
32X	<10 m	125 m

Tabell 6.2: Maksimal kabellengde ved forskjellige hastigheter

I link-laget benyttes 16 bit adresser kalt Local Identifier (LID). En LID er analog med MAC-adresser i Ethernet (se kapittel 5.1.1), da den bare brukes til switching internt i subnettet. Avsenders LID og mottakers LID er en del av Local Route Headeren (LRH) som finnes i alle pakker.

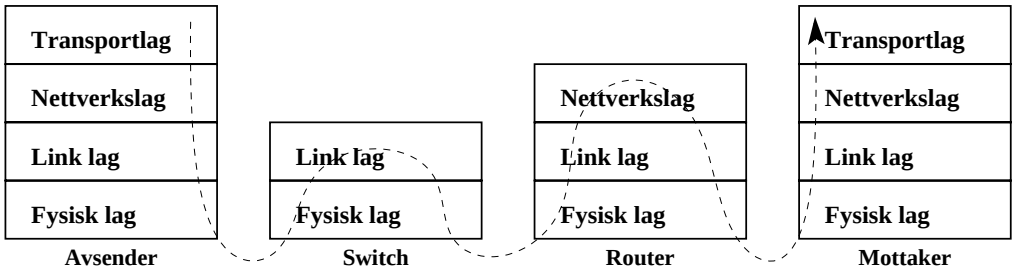
InfiniBand benytter prioriterte virtuelle kanaler (eng. *virtual lanes*, VL) for å støtte Quality of Service (QoS). VL15 brukes bare av managmentpakker og har høyest prioritet av alle kanalene. VL0 til VL14 kan brukes av applikasjoner, og høyere siffer betegner høyere prioritet. Pakken inneholder en Service Level (SL) som sier hvilken prioritet pakken skal ha gjennom nettet. Enheter i nettverket har egne tabeller for konvertering mellom SL og VL slik at QoS kan sikres over større nettverk med flere subnett.

Link-laget benytter *Credit Based Flow Control*. Denne flytkontrollen utføres ved at to enheter med direkte link mellom seg, utveksler credits. Flytkontroll foregår per virtuelle link, og en mottaker sender credits til en sender for fortelle hvor mye data den kan motta uten tap. På denne måten unngås sending av data dersom det ikke er plass i mottakers buffere.

Link-laget er også ansvarlig for å påse at pakker som mottas ikke er korrupte. Dette gjøres ved bruk av to CRC-koder per pakke. *Variant CRC* (VCRC) er på 16 bit og inkluderer alle felt i pakken. Den sjekkes og rekalkuleres ved hvert hopp i nettverket. *Invariant CRC* (ICRC) inkluderer bare de feltene som ikke endres når pakken beveger seg gjennom nettverket og har en størrelse 32 bit. Grunnen til at det benyttes to checksummer ligger i pålitelighet. Dersom en adapter har ustabilt minne, men beregner checksummen riktig, vil den kunne beregne riktig checksum for en ødelagt pakke. Dette vil ikke oppdages på videre hopp i nettverket ettersom checksummen er korrekt. Bruk av ICRC gjør at også slike feil kan oppdages.

6.2.3 Nettverkslag

Nettverkslaget håndterer routing av pakker mellom subnett og opererer ved bruk av en Global Route Header (GRH) i pakkene. Denne headeren inneholder 128 bits adresser, kalt Globally Unique Identifier (GUID), for mottaker og avsender. Routere i nettverket endrer mottakers LID i LRH på basis av subnett og GRH. GRH brukes ikke dersom mottaker og avsender ligger i samme



Figur 6.4: Lagdeling i InfiniBand

subnett. Figur 6.4 viser hvordan pakkene flyter mellom lagene i nettverket.

6.2.4 Transportlag

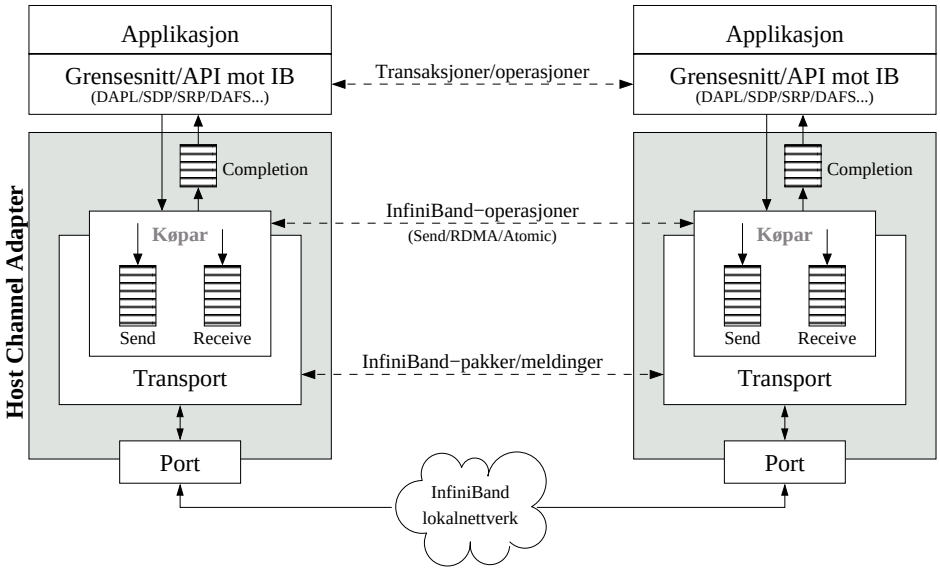
Et av de viktigste arkitektoniske elementene i InfiniBand-arkitekturen er køer. Applikasjoner bruker køer for å sende asynkrone forespørsler mot InfiniBand-hardwaren. Det finnes tre typer køer: *send*, *receive* og *completion*. Køene er organisert i køpar, eller *Queue Pairs* (QPs). Et køpar består av en kø for sending (*send*) og en kø for mottak (*receive*). En operasjon i en sendekø forteller hvilke data som skal sendes og hvor de skal. Operasjoner i mottakskøer forteller hvor data som mottas skal plasseres lokalt i minnet. Operasjonene utføres i den rekkefølge de ble lagt i køen. Alternativt kan også *completion*-køer benyttes slik at hardware kan varsle høyere lag når en operasjon er utført. Bruken av køer og køpar er illustrert i figur 6.5.

Sending av meldinger over InfiniBand på transportnivå utføres ved at meldingene sendes mellom to køpar. En forbindelse over InfiniBand opprettes ved at et køpar hos en enhet assosieres med et køpar hos en annen enhet. Dette oppnås ved at pakkene inkluderer et køpar-nummer som sier hvilken kø på en enhet de skal til. Tilsvarende angis et køpar-nummer ved bruk av forbindelsesløs kommunikasjon.

Køpar deles ikke mellom applikasjoner, og bruk av et køpar krever derfor ingen interaksjon mellom applikasjon og operativsystem. Opprettelsen av et køpar krever derimot at operativsystemet reserverer et minneområde køparet kan bruke som buffere for innkommende og utgående meldinger.

6.3 Kommunikasjon mellom prosesser

Køparene i InfiniBand støtter fem kommunikasjonsformer med forskjellig tjenestekvalitet:



Figur 6.5: Køpar og operasjoner mot InfiniBand-hardware (basert på figur 2 fra [Hav02b])

- **Reliable Connection (RC)** - Forbindelser settes opp mellom to enheter som skal kommunisere med hverandre. RC garanterer at meldinger kommer frem, mottas i riktig rekkefølge og er uten feil. Maksimal pakkestørrelse er ikke avhengig av pakkestørrelser på lavere nivå, men fragmentering og defragmentering av for store pakker for skjer i hardware.

- **Unreliable Connection (UC)** - Forbindelser settes opp mellom mellom to enheter som skal kommunisere med hverandre. UC har ingen garanti for at en pakke kommer frem. Pakker som inneholder feil sendes ikke automatisk på ny. UC garanterer likevel at dersom pakkene kommer frem, leveres de i riktig rekkefølge.

- **Unreliable Datagram (UD)** - Denne formen for kommunikasjon er forbindelsesløs. Den er upålitelig i form av at den ikke garanterer at meldinger kommer frem eller er i riktig rekkefølge. Den har heller ingen beskyttelse mot duplikate meldinger. Pakker som inneholder feil droppes, og pakker som ankommer høyere lag er derfor ikke korrupte. Pakkestørrelse ved bruk av UD er begrenset av pakkestørrelser i lavere lag.

- **Reliable Datagram (RD)** - Denne kommunikasjonsformen er også forbindelsesløs, men garanterer leveranse. Den har også garanti for at meldinger mottas i riktig rekkefølge og ikke dupliseres, og meldinger sendes automatisk på ny dersom de var korrupte ved mottak.

- **Raw Datagram** - Upålitelig transport der alle headere utenom LRH er fjernet. Denne formen for kommunikasjon kan brukes for å la pakker som benytter andre protokoller (f.eks. IP) overføres over InfiniBand.

Over disse kommunikasjonsformene tilbys tre typer operasjoner:

- **Meldingssending** - Meldingssending utføres med *send*- og *receive*-operasjonene. *Send* sender data i en lokal buffer til en annen node. Hvor i minnet hos mottaker de plasseres bestemmes av hvilke *receive*-buffere mottaker har "postet" til InfiniBand-adapteren.
- **RDMA** - Ved bruk av Remote Direct Memory Access (RDMA) kan en node lese og skrive direkte i forhåndsbestemte minnesegmenter hos en prosess på en annen node.
- **Atomiske operasjoner** - Atomiske operasjoner er RDMA-operasjoner som foretar lesing og skriving samtidig. Data sendes til mottakernode sammen med forespørselen. Ved *Compare and swap* inneholder forespørselen en sammenlikningsverdi og en bytteverdi. Dersom verdien i forespurt minneregion hos mottaker er lik sammenlikningsverdien, byttes den ut med bytteverdien. Ved *FetchAdd* inneholder forespørselen en verdi. Hos mottaker hentes ønsket minneverdi. Denne sendes tilbake til avsender. Hos mottaker legges verdien fra forespørselen til verdien fra minnet og summen overskriver gammel verdi i minnet hos mottaker.

6.4 Programmeringsgrensenitt

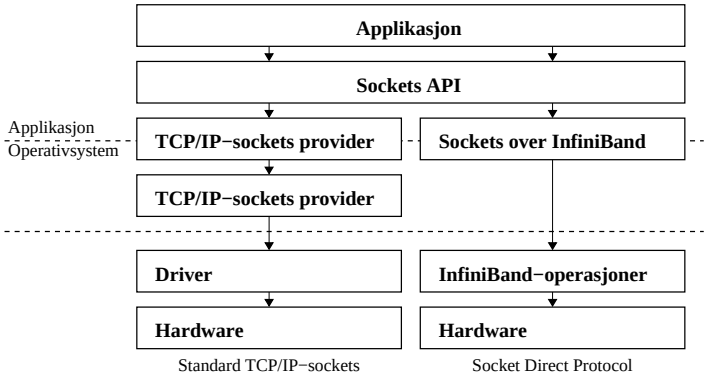
Det finnes flere protokoller og APIer for InfiniBand som kan benyttes for node til node kommunikasjon.

6.4.1 IPoIB - IP over InfiniBand

IPoIB (IP over InfiniBand) [IPI] er en standard som definerer hvordan IP-protokollen skal implementeres over InfiniBand. Den støtter støtter standard IP-relatert funksjonalitet som ARP og multicast. Ved bruk av IPoIB kan applikasjoner gjøre bruk av InfiniBand-hardware uten endring. IPoIB kan brukes sammen med SDP eller alene.

6.4.2 SDP - Socket Direct Protocol

SDP (Socket Direct Protocol) [SDP] muliggjør bruk av standard socket-API over InfiniBand. Sockets er en svært hyppig brukt API i form av for eksem-



Figur 6.6: Socket Direct Protocol (basert på figur fra [Mel00])

pel WinSock eller BSD-sockets, og SDP er implementert på en slik måte at applikasjoner skal kunne brukes uten særlig endring. SDP krever IPoIB for å fungere.

Figur 6.6 viser protokoll-stakken for SDP. Ved bruk av SDP over InfiniBand utføres en stor del av operasjonene i hardware, noe som reduserer CPU-bruken.

6.4.3 DAPL - Direct Access Programming Library

DAPL [DAP] er en transport- og plattformuavhengig API. Målet med DAPL er å tilby en standard API for programmering mot nettverksadaptere med støtte for RDMA. Den er inndelt i kDAPL (*kernel Direct Programming Library*) og uDAPL (*user Direct Programming Library*). kDAPL tilbyr RDMA-operasjoner for operativsystemets kjerne, mens uDAPL tilbyr RDMA-operasjoner med OS-bypass direkte fra applikasjonsområdet (*userspace*).

En grundigere beskrivelse av uDAPL gis i kapittel 8.

6.4.4 Andre APIer og protokoller

SRP (SCSI RDMA Protocol) [SRP] er en protokoll for overføring av SCSI-forespørsler mellom servere og lagringssystemer. Hovedmålene bak protokollen er høy ytelse og lav responstid, og protokollen benytter RDMA for å oppnå disse målene. ANSI T10 committee styrer utvikling av protokollen.

DAFS (Direct Access File System) [DAF] bruker RDMA for å oppnå høy ytelse og lav overhead ved aksess til delte filer. Ved bruk av DAFS kan filer fra en DAFS-server leses direkte inn i applikasjonsbuffere hos klient (OS-bypass). DAFS vedlikeholdes av DAFS collaborative.

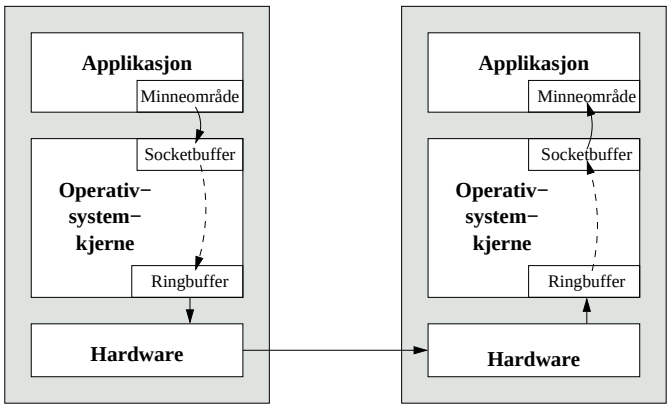
Kapittel 7

Hvorfor InfiniBand?

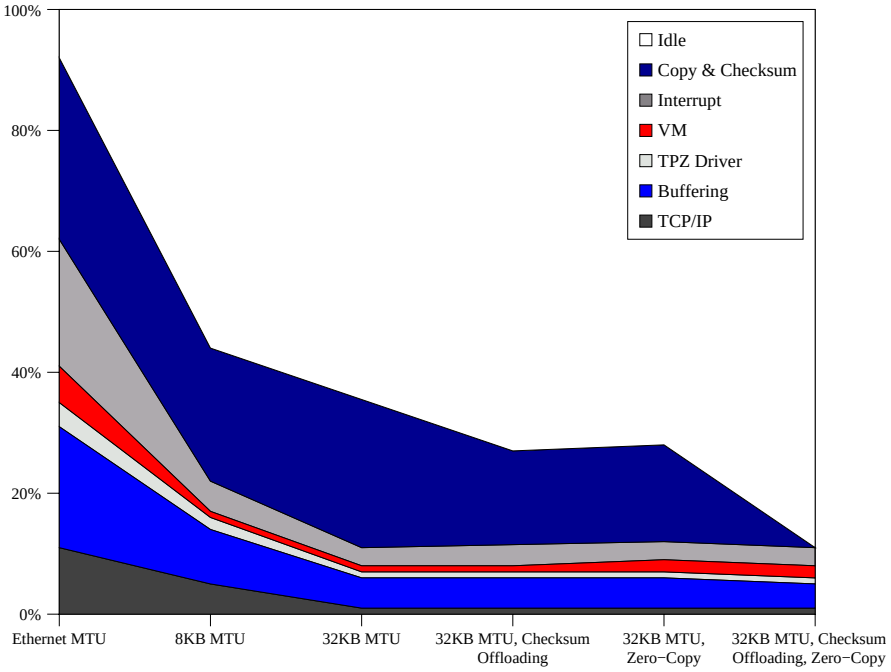
7.1 Hva er problemet med IP over Ethernet?

Ettersom Ethernet i dag tilbys med båndbredde helt opp i 10 Gbit/s, er overføringsraten mer enn høy nok, og transporttiden over lederen er også lav. Men sending og mottak av en IP-pakke over Ethernet innebærer en rekke operasjoner i software. Det vil her bli gitt en noe forenklet forklaring på utførelsen i Linux med kjerne 2.4 og bruk av sockets [Gou02]. Figur 7.1 illustrerer hvordan data flyter gjennom systemet ved sending og mottak.

Når applikasjonen kaller *send()*, kopieres data fra applikasjonens minneområde over i en buffer i operativsystemet. Data kopieres fordi *send()*-funksjonens semantikk tilsier at de data som var i minneområdet under kallet, er de data som skal sendes. Dersom applikasjonen endrer data i minneområdet like etter *send()*-kallet skal dette ikke reflekteres i sendte data. Da videre utførelse av sendingen skjer i operativsystemet, må det utføres et context-bytte. Et



Figur 7.1: Sending og mottak av en IP-pakke



Figur 7.2: Bidrag til CPU-bruk ved bruk av TCP/IP (Kilde: [GCY99])

context-bytte er et bytte av kjørende prosess og innebærer utskifting av data i registre på prosessor. Operativsystemet traverserer IP-stakken, og i transportlaget (UDP eller TCP) legges egne headere på pakken og checksum beregnes. Ved bruk av UDP kan denne checksum-beregningen slås av, noe som gir økt effektivitet, men mindre pålitelighet. IP-laget legger på en IP-header og beregner en checksum for denne headeren. Til slutt kopieres pakkens adresse (*copy by reference*) over til en ring-buffer som deles mellom driver og Ethernet-adapter (illustrert med stiplet linje i figur 7.1), og pakken kopieres til adapteren og sendes over lederen.

På mottakersiden mottas pakken fra Ethernet-hardwaren i en buffer i operativsystemet, og adressen legges i driverens ring-buffer for mottak. Adapterdriverens avbruddshåndterer gir operativsystemet beskjed om pakkeankomsten, og operativsystemet kontrollerer av checksummer, fjerner headere og kopierer til slutt bufferets adresse over i en mottakskø på riktig socket (stiplet pil i høyre del av figur 7.1). Når applikasjonen kaller *recv*, kopieres data fra kjernens buffer og over i minneområdet applikasjonen anga som parameter i *recv()*-kallet.

Som nevnt i kapittel 2.7, ligger store deler av problemet med IP i bruk av CPU. Undersøkelser [GCY99] utført ved Duke University viser at intern minne-til-minne-kopiering og beregning av checksum (CRC) er bakgrunnen for store deler av CPU-bruken. Figur 7.2 viser resultatene fra forsøket. Forsøket er riktignok utført over Myrinet (se kapittel 5.5) istedenfor Ethernet, men TCP/IP-stakken for de tre første verdiene på x-aksen er den samme. Overføringsraten

holdes konstant på 370 Mbit/s, mens pakkestørrelsen varieres. Sammenligning av verdiene for de tre pakkestørrelsene som bruker vanlig TCP/IP-stakk, viser at ved vanlig Ethernet-størrelse på pakkene gir kopiering og checksum, avbrudd, bufring og traversering av TCP/IP-stakken de største bidragene til CPU-bruken. Ved større pakkestørrelse senkes bidragene fra avbrudd, bufring og TCP/IP-stakken. Dette har blant annet sammenheng med at det sendes færre pakker, og dermed blir det også færre avbrudd. Samtidig viser grafen at bidraget fra kopiering og beregning av checksum holdes relativt konstant. Dette kan forklares ved at overføringsraten holdes konstant og at det dermed blir samme datamengde som skal kopieres.

Nyere Ethernet-adaptere har støtte for checksum-beregninger i hardware for å avlaste CPU for disse. Men minne-til-minne-kopiering er fortsatt et stort problem. Voltaire Inc. anslår i [Hav02a] at dersom en melding kopieres tre ganger (en gang fra adapter til minne og to ganger internt i minnet), vil et system med 133 Mhz SDRAM maksimalt kunne ha en overføringsrate 1,33 Gbit/s. Dette forutsetter 100% bruk av systembussen (noe som ikke er mulig), samt at det er sett bort ifra alle andre CPU-belastende operasjoner. Effektiv båndbredde blir dermed mye lavere. Som et eksempel forteller Voltaire videre at 200 Mbit/s trafikk i praksis bruker 30% av tilgjengelig CPU-kraft til rent nettverksarbeid.

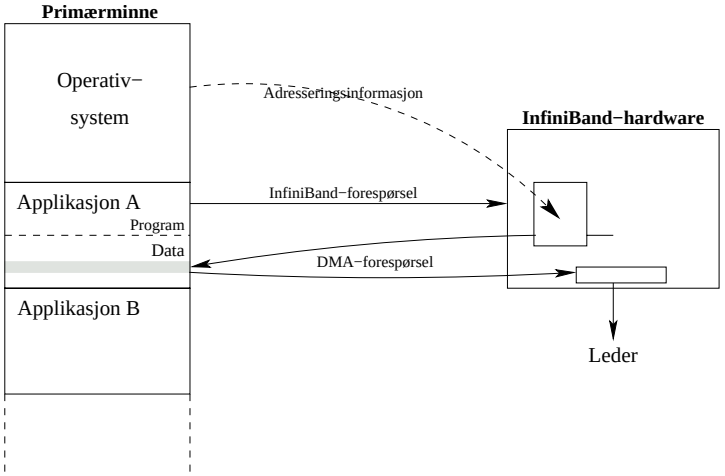
Kopieringen har også en effekt på prosessorens cache. Ettersom data går via CPU under kopieringen, vil cache på CPU i verste fall bli full av de data som kopieres. Dette medfører at CPU må aksessere primærminnet oftere like etter kopieringen.

Som nevnt medfører sending av pakken et context-bytte fra brukerapplikasjon til operativsystemet. Da dette krever utbytting av data i registre på CPU og lesing og skriving mot minnet, er dette en relativt dyr operasjon med tanke på CPU-bruk.

7.2 Hvordan løser InfiniBand dette?

For å redusere problemene forbundet med minne-til-minne-kopiering og unngå context-bytte til operativsystem, benytter InfiniBand seg av Direct Memory Access (DMA). InfiniBand-adapteren har en egen integrert DMA-kontroller, og ved bruk av denne kan InfiniBand-adapteren aksessere minnet uten interaksjon med CPU.

Context-bytte til operativsystemet unngås ved at applikasjoner kan sende meldinger direkte til og fra eget minneområde. For at dette skal fungere må noe minne reserveres InfiniBand-kontrolleren. Det er to årsaker til dette. Ved relativ adressering, vil ikke minneadresser brukt i en applikasjon peke på sam-



Figur 7.3: DMA og sending av data fra userspace

svarende fysiske adresse. Dette gjør minnehåndtering enklere, da applikasjoner kan swappes inn og ut av primærminnet uten at de nødvendigvis må ligge på samme fysiske minneadresse hver gang. Da InfiniBand-kontrolleren bare ser de fysiske minneadressene, må relative adresser oversettes til fysiske. Ved reservering av minne gir operativsystemet InfiniBand-adapteren nødvendig adresseringsinformasjon slik at DMA-kontrolleren kan oversette adresser den får fra applikasjonene. Samtidig låser operativsystemet minneområdet slik at det ikke swappes til disk. Bruken av DMA er illustrert i figur 7.3.

Den andre årsaken til reservering er av sikkerhetsmessige årsaker. En applikasjon skal ikke kunne aksessere minne hos andre applikasjoner. InfiniBand gjør bruk av *Protection Domains* (PD) for å skille forskjellige prosesser fra hverandre. Ressurser som benyttes av et QP er tilordnet et bestemt PD, og kontrolleren kan dermed utføre aksesskontroll for forespørslene fra applikasjonene.

Bruken av DMA gjør zero-copy-protokoller kan implementeres over InfiniBand. Ettersom DMA kan aksessere minnet direkte, trenger ikke data kopieres over i operativsystemets buffere.

7.3 Alternative løsninger for IP over Ethernet

Teknologiene beskrevet i kapittel 5 kan erstatte Ethernet i clusterløsninger, men det finnes også løsninger som gjenbruker IP- og Ethernet-teknologi. Fordelen med slike løsninger er at eksisterende applikasjoner kan brukes uten endring.

7.3.1 Zero-copy-implementasjon

Hovedfokus ved zero-copy-implementasjon er, som navnet tilsier, å kvitte seg med minne-til-minne-kopieringen.

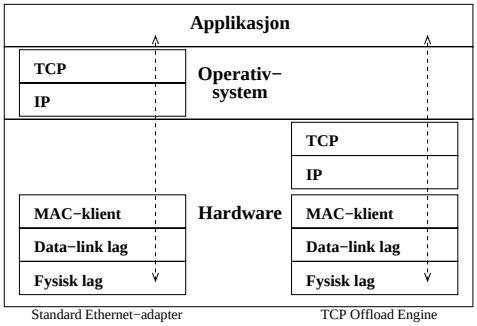
Trapeze/IP [GCY99] er et eksempel en zero-copy-variant av TCP/IP. Forsøket innebar endring av drivere og TCP/IP-stakk, men som tidligere nevnt ble denne testet over Myrinet istedenfor Ethernet. Figur 7.2 viser resultatene av dette forsøket. Ved beregning av checksum i hardware og bruk av zero-copy-protokoller, viser grafen at CPU-bruken kan senkes betraktelig (ca. 25%) ved sending av store meldinger.

I et annet forsøk [SPGH01], utført ved Universitet i Oslo, implementerte og vurderte deltakerne en zero-copy-versjon av UDP over Gigabit Ethernet. Dette ble gjort ved bruk av UVM i OpenBSD. UVM er et virtuelt minnesystem der applikasjon blant annet kan "låne" minnesider til operativsystemet.

7.3.2 TCP Offload Engine

Et annet alternativ er bruk av *TCP Offload Engine* (TOE) [YCM+02]. Motivasjon bak utviklingen av denne teknologien er færre avbrudd og reduksjon i CPU-bruk ettersom checksummer beregnes i hardware og færre byte kopieres over systembussen [Mog03]. For å oppnå dette implementerer en TOE TCP- og IP-lagene i hardware. Dette er illustrert i figur 7.4. Selv om bare TCP inkluderes i teknologinavnet, har implementasjonene ofte støtte for UDP.

Bruk av TOE krever egne drivere, men applikasjoner kan brukes uforandret. Ettersom all prosessering i IP-stakk er gjort før data når primærminnet, kan også antall kopieringer reduseres. Ved mottak kan det i beste fall oppnås en zero-copy-sti der data kopieres fra TOE og rett inn i applikasjonsbuffer.



Figur 7.4: Vanlig Ethernet-adapter og TCP Offload engine

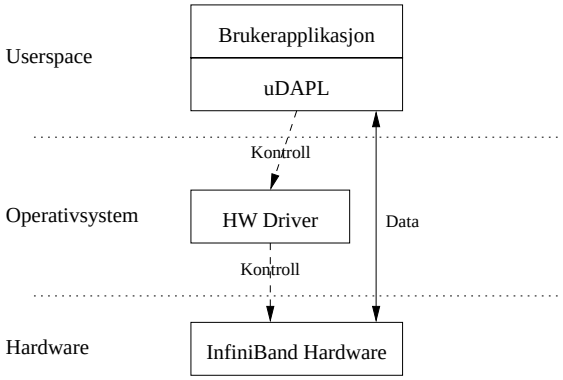
Kapittel 8

uDAPL

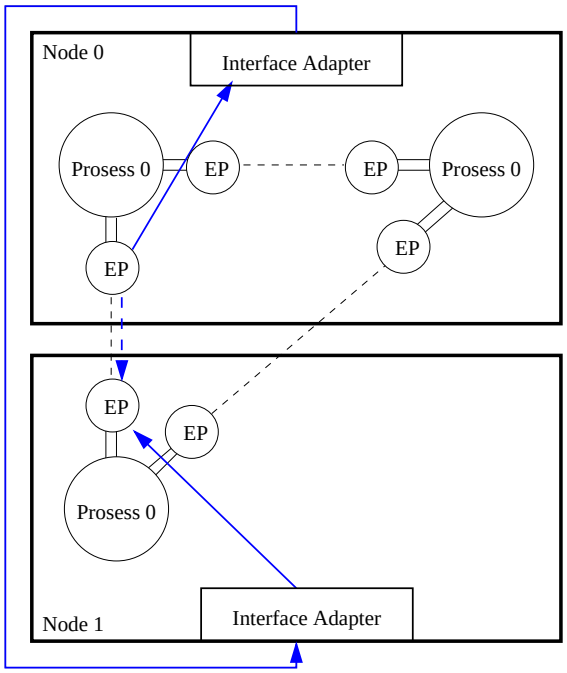
DAPL, eller *Direct Access Programming Library*, er som nevnt i kapittel 6.4.3 en standardisert plattformuavhengig API for programmering mot nettverksadaptere med støtte for *Remote Direct Memory Access* (RDMA). DAPL er utviklet av DAT Collaborative og finnes i to varianter: kDAPL og uDAPL. *Kernel Direct Access Programming Library* (kDAPL), tilbyr en RDMA-API for operativsystemet, mens *User Direct Access Programming Library* (uDAPL) tilbyr RDMA direkte til brukerapplikasjoner. Nyeste versjon for uDAPL var versjon 1.1 da dette ble skrevet.

8.1 Arkitektur

Figur 8.1 viser kontroll- og dataflyt i uDAPL. Kontrollmeldinger går til hardwaren via driveren i operativsystemet, mens data kopieres direkte fra brukerapplikasjonens minneområde til hardwaren. Ved bruk av InfiniBand gjøres datakopieringen ved bruk av DMA, og operasjonene bruker dermed svært lite



Figur 8.1: Kontroll og dataflyt i uDAPL



Figur 8.2: Endpoints og dataflyt i uDAPL

CPU-kraft.

Ettersom uDAPL er uavhengig av hvilken RDMA-kompatibel nettverksadapter som benyttes, brukes *Interface Adapter* som en abstraksjon for denne.

8.1.1 Forbindelsesmodell

Meldinger i uDAPL sendes fra et *endpoint* til et annet. Et endpoint representerer et endepunkt i en forbindelse, og har mange likheter med en socket i sockets-APIen. Figur 8.2 viser prosesser som er knyttet opp mot hverandre i uDAPL. De stiplede sorte linjene representerer forbindelser. Den stiplede blå pila viser konseptuell dataflyt mellom to prosesser, mens de heltrukne blå pilene viser hvordan data faktisk flyter mellom prosessene. Ettersom uDAPL er forbindelsesorientert vil det være ett endpoint hos hver av de to kommuniserende prosessene for hver forbindelse. En prosess kan ha flere endpoints mot samme interface-adapter og flere prosesser kan ha tilgang til samme interface-adapter.

For å lytte etter tilkoblingsforespørsler brukes *Reserved* eller *Public Service Points* (RSP eller PSP). Når en prosess skal koble seg til en annen, lager den et endpoint og sender en tilkoblingsforespørsel til et service-point hos prosessen den skal koble seg til. Dersom prosessen godtar tilkoblingen, lager den et eget endpoint og etablerer en forbindelse mellom de to prosessene. En RSP lytter og oppretter bare én forbindelse, mens en PSP kan motta flere forbindelser.

8.1.2 Hendelsesmodell

Svar på forespørsler fra et endpoint mot en interface-adapter gis i form av asynkrone hendelser. Hendelser er organisert i hendelsesstrømmer og disse kan samles i hendelseskøer ved bruk av *Event Dispatchers* (EVD). Et endpoint kan tilknyttes tre ulike hendelseskøer: en for tilkoblingsrelaterte hendelser (connect evd), en for sending av data (request evd) og en for mottak (receive evd).

En prosess kan polle eller bruke blokkerende venting i påvente av resultat i en hendelseskø. I tillegg kan prosessen gjøre bruk av *Consumer Notification Objects* (CNO). En CNO muliggjør polling eller blokkerende venting på flere hendelseskøer samtidig.

8.1.3 Dataoverføring

uDAPL definerer to par dataoverføringsoperasjoner. *RDMA-read* og *RDMA-write* brukes for å lese og skrive direkte i et forhåndsdefinert minneområde hos en annen prosess.

Ved bruk av *send* og *receive*, benytter prosessene buffere for meldingene. En prosess som skal motta meldinger "poster" (tilgjengeliggjør) buffere for mottak. En prosess som sender data, poster bufferen med data, og data kan dermed flyttes direkte fra den sendende prosessens buffer til en av mottaksprosessens buffere. Ettersom *send* utfører meldingssending asynkront ved bruk av en kø og hendelser, har *send* en annen semantikk en tilsvarende funksjon ved bruk av sockets. Ved bruk av *send*-kallet i sockets garanteres det at de data som lå i bufferen da funksjonen ble kalt, er de data som sendes. Dette kan garanteres ettersom kallet medfører at data kopieres over i operativsystemets minneområde. Ved bruk av uDAPL kopieres data ut på et senere tidspunkt, og det kan ikke gis en slik garanti. Applikasjonen må av den grunn ikke endre data i bufferen før den har mottatt en hendelse om at sendingen er utført.

8.1.4 Minnehåndtering

Som beskrevet i kapittel 7.2 må minnesegmenter registreres med nettverksadapteren. I uDAPL defineres to typer minneregioner: LMR og RMR. LMR, eller *Local Memory Region*, er lokalt minne som er registrert i nettverksadapteren. Dette minnet kan være destinasjon for RDMA-read eller en receive-operasjon, eller det kan være kilden til en RDMA-write eller en send-operasjon.

RMR, eller *Remote Memory Region*, er minne hos en annen prosess som brukes ved RDMA. Dersom en prosess skal benytte RDMA, må den binde en RMR til en LMR hos den andre prosessen. Flere RMR kan være bundet til samme LMR, slik at flere prosesser kan skrive eller lese fra samme minneområde.

Ved binding av RMR til LMR produseres en nøkkel som brukes i RDMA-operasjonene. Bruken av nøkkel muliggjør autorisering, ettersom nøkkelen må brukes for å få tilgang til minnet.

For å beskytte ressurser hos adapter fra andre prosesser på samme maskin, tilordnes elementene en Protection Zone (PZ). Når uDAPL-operasjoner skal utføres, sjekkes det om de involverte elementene tilhører samme PZ. Dermed kan ressursene beskyttes mot feilende eller ondsinnede applikasjoner.

8.1.5 Adressering

uDAPL-spesifikasjonen benytter IP-adresser (IPv4 eller IPv6) for adressering. Ettersom flere service-points kan åpnes på samme node, brukes en *Connection Identifier*, som er en unik identifikator for hvert service-point. Denne kan sammenliknes med portnummer ved bruk av IP og brukes for å angi hvilket service-point en prosess ønsker å koble seg til. På denne måten kan service-points fra samme eller andre prosesser skilles fra hverandre.

8.2 uDAPL over InfiniBand

Ved bruk av uDAPL over InfiniBand knyttes et endpoint til et køpar (se kapittel 6.2.4). Operasjoner fra uDAPL oversettes til InfiniBand-operasjoner og legges i køene. Tilsvarende genereres hendelser i uDAPL-køene utifra hendelser fra completion-køen i InfiniBand. Et endpoint i uDAPL benytter som nevnt en *Protection Zone* for beskyttelse av ressurser og denne er direkte tilknyttet et *Protection Domain* i InfiniBand.

For adressering bruker uDAPL IP over InfiniBand for å finne riktig InfiniBand-GID utifra IP-adressen. Dersom IP over InfiniBand ikke er tilgjengelig kan GID angis sammen med hostnavn i en konfigurasjonsfil.

Kapittel 9

Ytelsesmåling

For å undersøke forskjeller i CPU-bruk og ytelse, ble det skrevet tre par testapplikasjoner. Dette kapitlet presenterer applikasjonene og sammenligner resultater fra kjøring over Gigabit Ethernet og 4X InfiniBand.

9.1 Applikasjoner

Hvert par består av en meldingssender og en meldingsmottaker, der meldingssender sender meldinger av bestemte størrelser til mottaker i hurtig tempo. Hver melding består av tilfeldige tall og bokstaver og endres noe mellom hver sending. Antall meldinger som skal sendes angis som en parameter til meldingssender.

Alle programmene registrerer mottatt/sendt datamengde og tid fra første til siste melding.

9.1.1 UDP

Applikasjonsparet for sending av meldinger over UDP består av udpsend og udpreceiver. Sistnevnte lytter på en forhåndsbestemt port, mens udpsend sender meldinger til denne.

For å måle ytelse, er det ønskelig å registrere hvor lang tid det tar fra første til siste melding sendes eller mottas. For sending er tidtakingen triviell, men for mottak medfører bruk av UDP noen små implikasjoner. Ettersom UDP er upålitelig, kan pakker forsvinne. Dette kan blant annet skje dersom mottaker ikke klarer å motta meldinger like hurtig som de sendes. Dette medfører at pakkeantall eller meldingsinnhold ikke kan brukes i tidtakingen ad hoc. Samtidig er UDP forbindelsesløs, og dermed kan heller ikke lukking av forbindelsen brukes som en stopper for tidtakingen.

I udpreceiver startes tidtakingen når den mottar første melding. For å kunne stoppe tidtakingen, kontrollerer udpreceiver meldingsinnholdet i mottatte meldinger. Som siste melding, sender udpsend en melding med et bestemt innhold ("Message: LAST"). Når udpreceiver mottar denne stoppes tidtakingen. Ettersom meldinger kan forsvinne, sender udpsend denne meldingen tre ganger.

9.1.2 TCP

Ettersom TCP også benytter IP over Ethernet, ble det skrevet et applikasjonspar også for denne protokollen. Tcpreceiver lytter først etter en innkommende tilkoblingsforespørsel på en bestemt port, og mottar deretter meldinger over denne forbindelsen. Avsender, tcpsend, kobler seg tilsvarende til tcpreceiver og sender meldinger.

Tidtaking hos tcpreceiver startes etter at forbindelsen er opprettet, og avsluttes når den mottar "Message: LAST". Tidtaking hos tcpsend startes før sending av første datamelding og stoppes etter sending av siste.

9.1.3 uDAPL

Applikasjonsparet for uDAPL består av ibsend og ibreceiver. Ibreceiver oppretter først et service-point der den lytter etter innkommende tilkoblingsforespørsler. Etter opprettelse av en tilkobling, stiller den seg så i mottaksmodus. Ibsend kobler seg tilsvarende til og sender meldinger.

Ettersom Sun HADB er et shared-nothing databasesystem, ble send- og receive-kallene valgt fremfor RDMA-read og RDMA-write. Bakgrunnen for dette var at send og receive har en semantikk som er mer lik kallene fra sockets-APIen som allerede brukes i systemet i dag. Samtidig forutsetter de ikke at en sendende prosess har kjennskap til hvordan mottakende prosess organiserer sine meldingsbuffere.

For å kunne utnytte den asynkrone formen for meldingssending som benyttes i uDAPL, blir det opprettet en rekke buffere hos ibreceiver og ibsend. Ibsend har et fast antall buffere som den benytter ved sending av meldinger. Bufferne ligger i en sirkulær liste, og ved sending av en melding sjekkes det om neste buffer er ledig. Dersom den er det, markeres den som opptatt, endres og send-kallet *dat_post_send()* utføres. Dersom den ikke er ledig, sjekkes det om resultater fra tidligere send-kall er tilgjengelige i hendelseskøen. Buffere som har blitt sendt, markeres som ledige og kan brukes til videre sending. Pseudokode for dette finnes i pseudokode 9.1.

På tilsvarende måte har ibreceiver et tilsvarende antall buffere. Disse tilgjengeliggjøres InfiniBand-hardwaren gjennom kallet *dat_post_recv()*. Deretter utfører

```
Function sendMessage()
 1: nextBuf = (nextBuf + 1) % antall buffere
 2: while (buffer[nextBuf] is busy) do
 3:    for all completed send events do
 4:       Mark buffer from event as free
 5:    end for
 6: end while
 7: Mark buffer[nextBuf] as busy
 8: Randomly change parts of buffer[nextBuf]
 9: dat_post_send(buffer[nextBuf])
```

Pseudokode 9.1: Pseudokode for sending i ibsend

mottaker blokkerende vent på hendelseskøen for mottak. Når den mottar en hendelse, kontrolleres tilsvarende buffer, og deretter kalles *dat_post_recv()* for å tilgjengeliggjøre bufferen for en ny melding.

På samme måte som for tcpreceiver, startes tidtaking i ibreceiver like etter at forbindelsen er opprettet og stoppes når forbindelsen lukkes. For ibsend startes tidtakingen før sending av første datamelding, og lukkes like etter sending av siste.

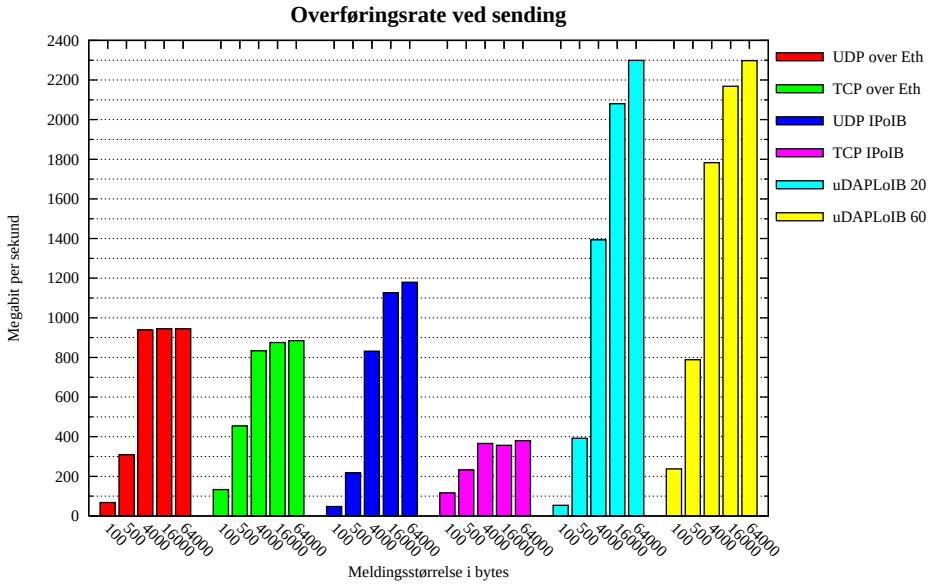
9.2 Testutførelse, hardware og støtteprogrammer

Testene ble utført på to Sun Fire v210 utstyrt med to UltraSparcIII 1 GHz prosessorer hver og 2048 MB RAM. Serverne benyttet en prerelease av Solaris 10 (SunOS release 5.10). Som vanlig ved bruk av en prerelease, oppstod det enkelte problemer. En av problemene innebar blant annet at uDAPL-kode fikk systemet til å gå i vranglås. Men etter noen uker med prøving, feiling og kontakt med utviklere i USA, ble problemene løst og systemet stabilt.

For kommunikasjon hadde serverne både Gigabit Ethernet og 4X InfiniBand. InfiniBand-kortene var PCI-X-kort av typen Mellanox Tavor Jaguar MT23108, og disse var knyttet sammen via en Sleipnir-switch. I testene ble bare en av to kanaler på kortene tatt i bruk.

Vmstat ble benyttet for registrering av CPU-bruk. Vmstat kjører som en egen prosess og registrerer CPU-bruk for system- og brukerprosesser ved forhåndsbestemt intervaller. I testene ble vmstat satt til å registrere CPU-bruk en gang per sekund.

For hver av de tre applikasjonsparene, ble det kjørt tester med fem forskjellige meldingsstørrelser: 100 byte, 500 byte, 4000 byte 16 000 byte og 64 000 byte. For de forskjellige meldingsstørrelsene ble det sendt henholdsvis 16 millioner, 4 millioner, 1 million, 500 000 og 250 000 meldinger.



Graf 9.1: Overføringsrate ved sending av meldinger

Testene for uDAPL ble kjørt mot to varianter av uDAPL-programmene. I den ene varianten, heretter kalt uDAPL 20, ble det benyttet 20 buffere på 64 000 byte, mens det i den andre varianten, heretter kalt uDAPL 60, var 60 buffere på 64 000 byte. Versjon 1.1 av uDAPL ble benyttet i uDAPL-testene.

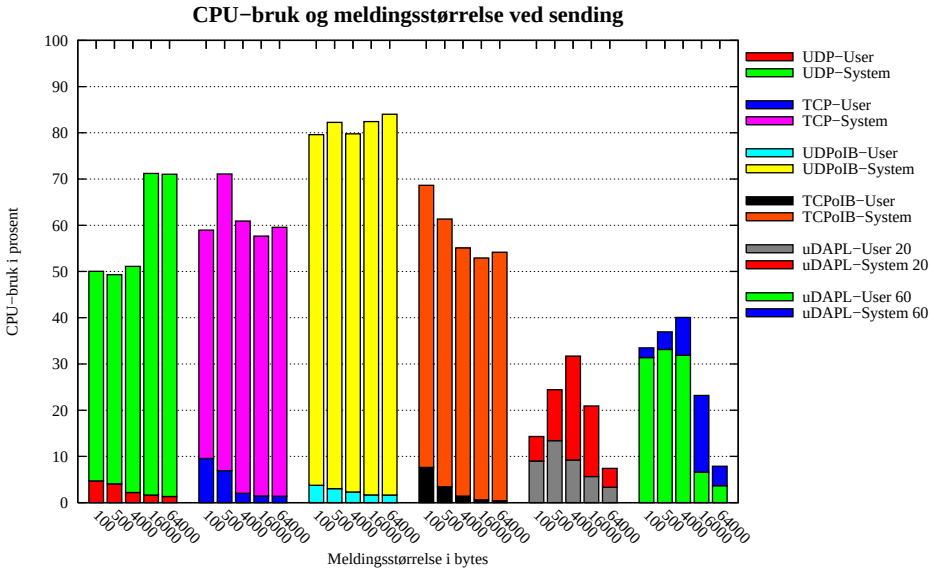
Ettersom også IP over InfiniBand (kapittel 6.4) var tilgjengelig, ble applikasjonsparene for UDP og TCP også kjørt over denne IP-versjonen.

9.3 Resultater

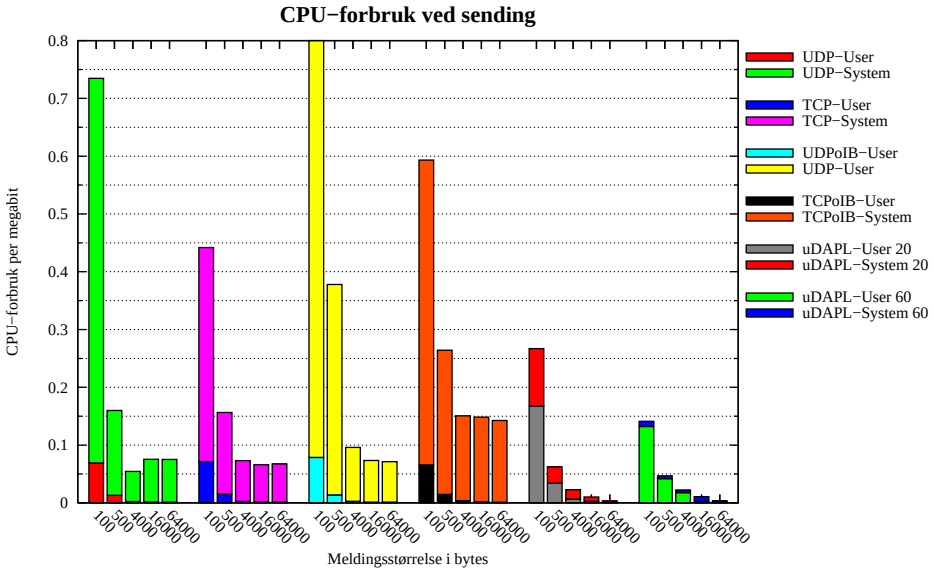
Data fra kjøring ble samlet inn for både meldingssender og meldingsmottaker for de forskjellige testkjøringene. Resultater fra testene er beskrevet i de to neste delkapitlene.

9.3.1 Meldingssending

Graf 9.1 viser gjennomsnittlig overføringsrate ved sending av meldinger i de fire forskjellige meldingsstørrelsene. Ved 100 og 500 meldingsstørrelse er overføringsraten faktisk større for TCP over Ethernet enn for uDAPL 20, mens uDAPL 60 nesten dobler ytelsen for TCP. uDAPL 20 blir merkbart bedre ved økende meldingsstørrelse. Ved en meldingsstørrelse på 4000 byte viser grafen at uDAPL 60 har over dobbelt så høy overføringssrate som IP-variantene, mens uDAPL 20 vedlikeholder denne forskjellen for størrelser fra 16 000 og 64 000 byte.



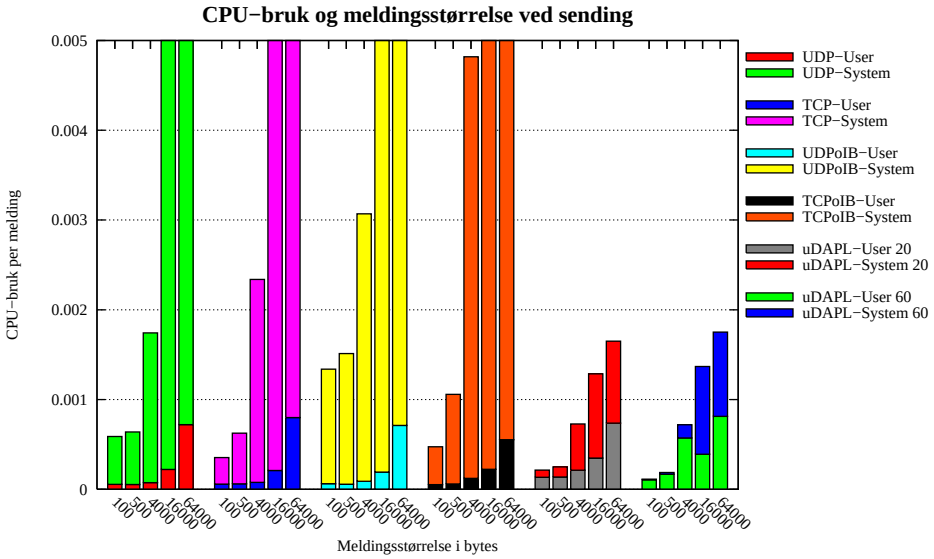
Graf 9.2: CPU-forbruk ved sending av meldinger



Graf 9.3: CPU-forbruk per megabit ved sending av meldinger

Gjennomsnittlig CPU-bruk ved sending av meldinger er illustrert i graf 9.2, og 100% betyr full last på begge prosessorer. Grafen viser at uDAPL har mindre CPU-bruk enn IP-variantene. Grafen viser også at en større andel av CPU-tiden i uDAPL brukes i brukerapplikasjonen, mens operativsystemet står for store deler av CPU-bruken ved IP-variantene.

Graf 9.2 gir likevel et noe uriktig bilde av CPU-bruken ettersom uDAPL har høyere overføringsrate enn IP-variantene. I graf 9.3 vises gjennomsnittlig CPU-bruk per megabit som overføres. Her kommer det tydelig frem at uDAPL har



Graf 9.4: CPU-forbruk per melding ved sending

mindre CPU-bruk enn IP-variantene. Ved meldingsstørrelser fra 500 byte og oppover er CPU-lasten mindre enn halvert ved bruk av uDAPL sammenlignet med IP-variantene. Graf 9.3 er kuttet for å får frem forskjellene ved høyere meldingsstørrelser. Den er gjengitt uten beskjæring som graf A.1 i vedlegg A.

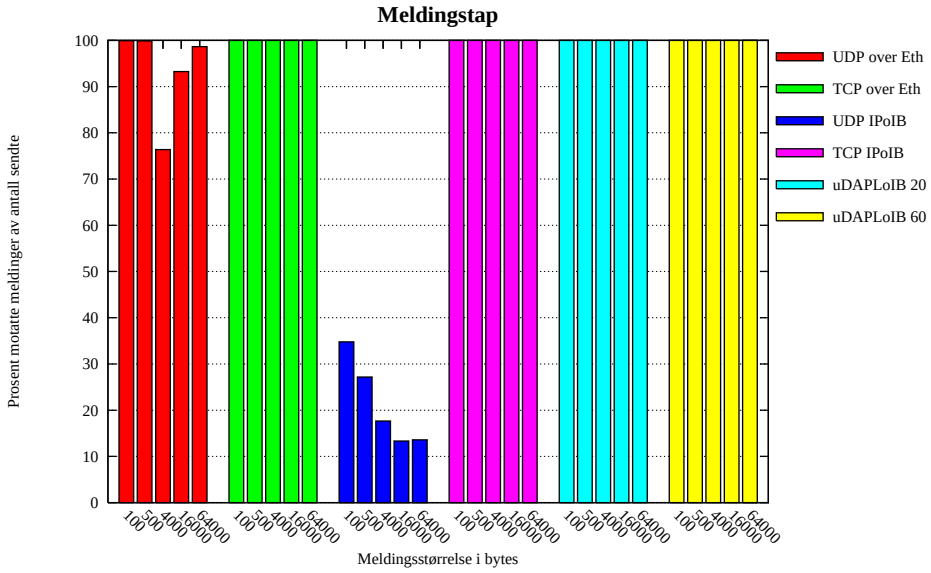
Graf 9.4 viser på tilsvarende måte CPU-forbruk per sendte melding. Også denne grafen er noe kuttet for å få frem forskjeller ved bruk av uDAPL, og den er gjengitt uten beskjæring som graf A.2 i vedlegg A. Grafen viser at uDAPL-variantene har lavere CPU-bruk per melding, noe som spesielt gjør seg gjeldene ved høy meldingsstørrelse.

9.3.2 Meldingsmottak

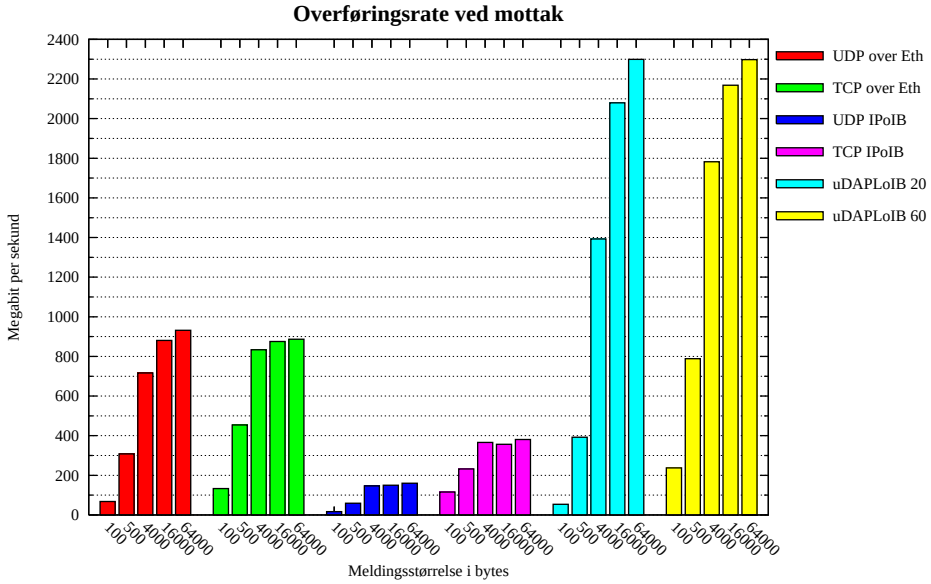
Graf 9.5 viser hvor stor andel av de sendte meldingene som mottas av mottakerdelen av applikasjonsparet. Ettersom både uDAPL og TCP er pålitelige forbindelsesorienterte protokoller, mottas som ventet alle sendte meldinger av meldingsmottaker. Ved bruk av UDP mistes derimot mange meldinger når meldingene blir store. Dette er spesielt tydelig ved bruk av UDP og IP over InfiniBand der over halvparten av meldingene ikke mottas.

Forskjeller i overføringsrate ved mottak er illustrert i figur 9.6. For TCP og uDAPL vises samme trend som ved sending av meldinger. For UDP og IP over InfiniBand er det derimot stor forskjell mellom sending og mottak. Dette skyldes, som vist i graf 9.5, at en veldig stor del av meldingene aldri mottas av mottaker.

Graf 9.7 viser gjennomsnittlig CPU-bruk målt med vmstat. Som ved sending



Graf 9.5: Antall mottatte pakker i prosent av antall sendte

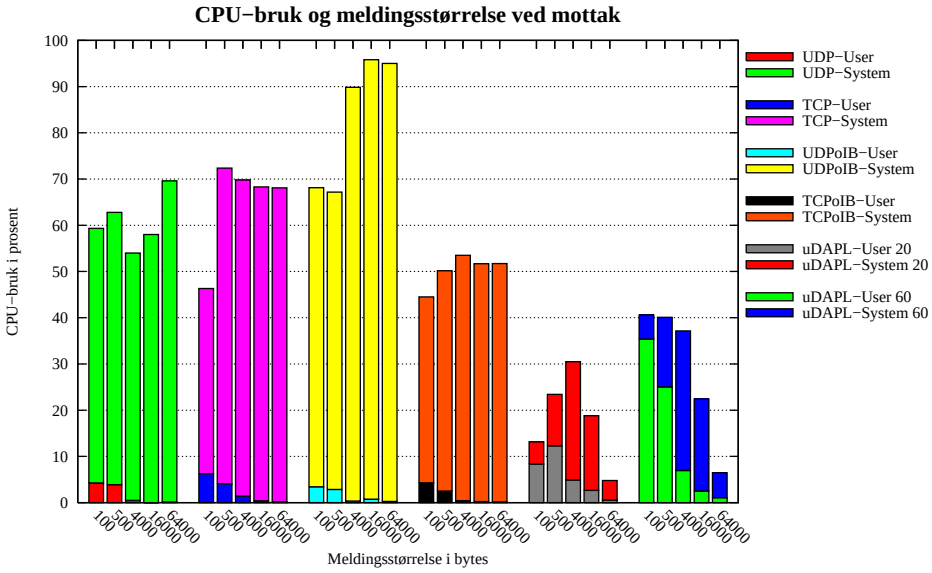


Graf 9.6: Overføringsrate ved forskjellige meldingsstørrelser

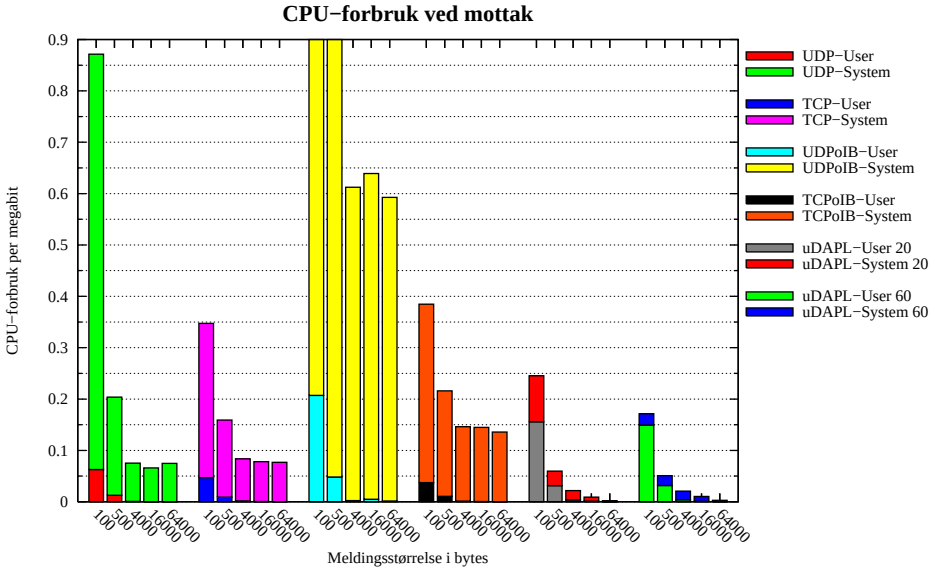
av meldinger, belaster uDAPL CPU mye mindre enn IP-variantene.

På samme måte som graf 9.3, viser graf 9.8 CPU-bruk dividert med overføringsrate. Resultatene viser at uDAPL bruker mye mindre CPU per mottatte megabit enn IP-protokollene når meldingsstørrelsen blir stor. Grafen er noe kuttet for bedre å illustrere forskjellen for store meldingsstørrelser. Den er gjengitt uten beskjæring som graf A.3 i vedlegg A.

Graf 9.9 viser CPU-forbruk per mottatte melding. Grafen er noe beskjært for

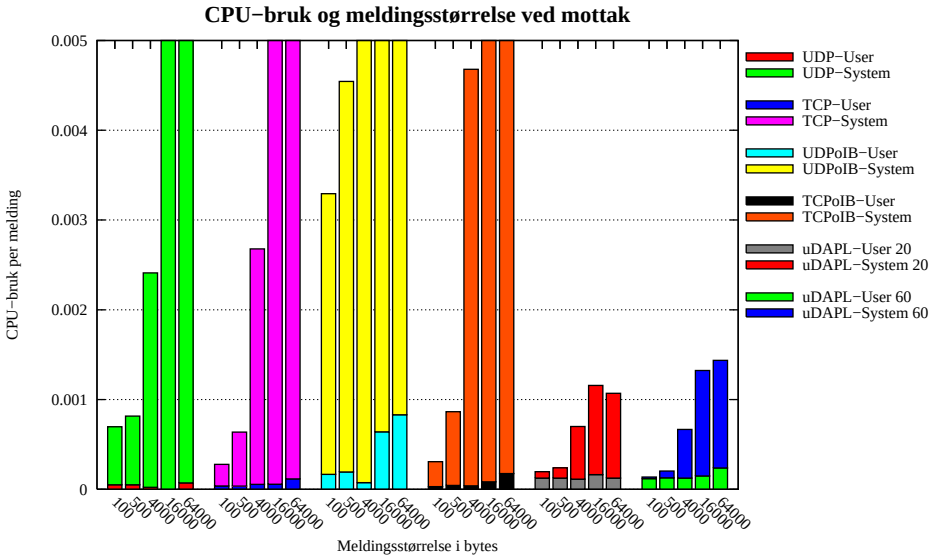


Graf 9.7: CPU-forbruk ved mottak av meldinger



Graf 9.8: CPU-forbruk per megabit ved mottak av meldinger

å vise forskjeller ved mottak meldinger i forskjellig størrelse i uDAPL. Den er gjengitt uten beskjæring som graf A.4 i vedlegg A. Også her viser grafen at uDAPL har mye lavere CPU-bruk per melding, noe som gjør seg sterkt gjeldende ved mottak av store meldinger.



Graf 9.9: CPU-forbruk per melding ved mottak

9.4 Diskusjon og feilkilder

Resultatene viser at bruk av uDAPL og InfiniBand både kan redusere CPU-bruk og øke overføringsraten samtidig sammenlignet med TCP og UDP over Ethernet. Graf 9.2 og 9.7 viser at CPU-forbruket øker når meldingene blir større ved bruk av TCP og UDP. For uDAPL over InfiniBand synker derimot CPU-forbruket ettersom det sendes færre meldinger. Dette skyldes at CPU-forbruket for håndtering av en melding i stor grad bare er knyttet til håndtering av buffere og sending av forespørsler mot hardware.

Sammenligning av uDAPL 20 med uDAPL 60 i graf 9.2 og 9.7 viser at ved få buffere brukes en større del av CPU-tiden i operativsystemet. Dette kan skyldes at applikasjonen i større grad må vente på at buffere skal bli ledige. Ventingen utføres ved blokkerende operasjoner som involverer operativsystemet. Sammenligning av CPU-tid ved sending og mottak av 4000 byte meldinger og bruk av uDAPL viser at ved mottak involveres operativsystemet i større grad enn ved sending. Dette kan skyldes blokkerende operasjoner hos mottaker, mens det hos sender også er noe CPU-bruk for endring av buffer før sending.

Graf 9.9 illustrerer hvordan CPU-forbruk per melding påvirkes av meldingsstørrelsen ved bruk av uDAPL. CPU brukt i userspace er relativt lik ved meldingsstørrelsene, mens CPU-tid brukt operativsystemet øker med meldingsstørrelsen. Dette skyldes at CPU-kostnad for å poste en mottaksbuffer er uavhengig av størrelsen. Tid brukt i operativsystemet øker fordi applikasjonen utfører blokkerende venting oftere i påvente av at en melding skal overføres til en buffer av InfiniBand-maskinvaren.

Ved UDP og TCP over Ethernet viser graf 9.1 og 9.6 at økning i overføringsrate flater ut når meldingene blir større enn 4000 byte. Ettersom CPU ikke er 100% i bruk tyder dette på at programmene har ligget like i underkant av det Ethernet-maskinvaren maksimalt klarer å overføre.

Graf 9.2 og 9.7 viser også en egenskap ved operativsystemet ved bruk av TCP og UDP. Applikasjonene er i utgangspunktet enkelttrådede og vil derfor bare kjøre på en av de to CPUene i hver maskin. Men ettersom TCP og UDP bruker over 50% av CPU-kraften på maskinene, bruker operativsystemet også den andre CPUen til deler av IP-utførelsen.

Det må også nevnes at vmstat rapporterer CPU-bruk for alle prosesser på datamaskinen. Prosesser eller operativsystemoperasjoner som i utgangspunktet er irrelevante for testen kan dermed påvirke CPU-målingene. Men ettersom det ikke ble kjørt andre CPU-krevende prosesser på serverne og det ble sendt mange meldinger, forventes det at slik CPU-bruk vil være omtrent lik for de forskjellige testene og ikke påvirke gjennomsnittsresultatene nevneverdig.

Sammenlignes TCP og UDP over Ethernet med samme protokoll over IP over InfiniBand, viser resultatene at IP over InfiniBand generelt sett yter dårligere. Det eneste unntaket er TCP over InfiniBand i graf 9.7, men dette er noe misvisende ettersom TCP over Ethernet har høyere overføringsrate enn TCP over InfiniBand. Dette kommer også frem av graf 9.8. Samtaler med utviklere hos Sun, antyder at ytelsessenkingen for IP over InfiniBand har sammenheng med FireEngine i Solaris 10. FireEngine er navnet på Solaris IP-implementasjon, og denne inneholder flere optimaliseringer for bruk mot Ethernet. En av disse forbedringene innebærer at checksum-beregninger for IP-pakker forventes utført i nettverksadapter. Ettersom InfiniBand-kortene ikke støtter checksumberegning for IP, må dette gjøres i operativsystemet.

Et av de mest oppsiktsvekkende resultatende er meldingstapet for UDP over InfiniBand i graf 9.5. Ved store meldinger mottar udpreceiver bare litt over 10 prosent av meldingene. Sammenligning av graf 9.6 og 9.1, viser tilsvarende at meldingene sendes med mye høyere rate enn de mottas. Dette kan skyldes problemer med tidligere nevnte FireEngine og mottak av større UDP-meldinger.

Kapittel 10

InfiniBand i Sun HADB

Dette kapitlet gir en kort beskrivelse av Sun HADB og forklarer hvordan InfiniBand ble innført. Til slutt gis resultater og en diskusjon rundt disse.

10.1 Sun High-Availability DataBase

Sun HADB, eller Clustra som den het tidligere, er et parallelt shared-nothing databasesystem. Det er designet for høy tilgjengelighet og skal maksimalt ha 5 minutter nedetid per år. Systemet tilbyr høy ytelse som skalerer ved endring av antall noder systemet består av.

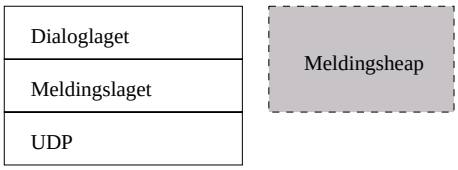
Det gis ikke en fullverdig beskrivelse av HADB her. Beskrivelsen dekker bare de deler som er relevante for implementasjonen og testingen.

10.2 Prosesser i HADB

Hver node i HADB består av fem prosesser. Disse tar seg av de forskjellige operasjonene hver node utfører.

Prosessen **clu_nsup_srv** overvåker de andre prosessene og starter dem på nytt dersom de skulle falle bort. Det er også denne som tar seg av *I'm alive*-meldinger mot andre noder.

Utførelse av transaksjoner styres av **clu_trans_srv**. Denne prosessen inneholder blant annet trådene TCON, UCHN og KERN. TCON er ansvarlig for forbindelser mot klienter og kontroll av klientenes transaksjoner. UCHN er ansvarlig for å lese logg og oversende logginnslag til speilnoden. KERN er databasens kjerne og er den tråden som utfører operasjoner på poster. Den tar seg også av vanlig databaselogg og lagring til disk.



Figur 10.1: Lagdeling for kommunikasjon i HADB (basert på figur 13 i [MS00])

SQL-forespørsler håndteres av **clu_sql_srv**, mens relasjonsalgebra som join, divisjon og differanse håndteres av **clu_relalg_srv**. Bruk av delt minne på en node styres av **clu_sqlshm_srv**.

10.3 Dagens kommunikasjonssystem

Kommunikasjonssystemet i HADB er inndelt i to lag: meldingslaget og dialoglaget (se figur 10.1).

10.3.1 Dialoglag

Dialoglaget håndterer dialoger mellom prosesser. En dialog er en rekke meldinger som representerer en virtuell RPC-liknende kommunikasjonslink mellom to prosesser eller en prosess og en klient. En dialog er pålitelig i form av at meldinger kommer i riktig rekkefølge og er feilfrie. Dialoglaget bufrer også utgående meldinger.

10.3.2 Meldingslag

Meldingslaget tilbyr funksjoner for transport av meldinger til dialoglaget. I praksis vil dette si at meldingslaget abstraherer bort for overliggende lag hvilken protokoll som benyttes. I dagens HADB brukes sockets over UDP/IP. Prosessene lytter på portnumre definert i konfigurasjonsfila, og IP-adresse og portnummer benyttes for å identifisere hvilken motpart en prosess kommuniserer med.

For sending av meldinger tilbyr meldingslaget funksjonen *clu_sendMessage()*. Denne funksjonen kaller *sendMessageUDP()* som gjennom noen mellomliggende kall kaller socket-operasjonen *sendto()*, som da foretar sending av meldingen som beskrevet for *send()* i kapittel 7.1.

For mottak av meldinger tilbys funksjonen *clu_recvMessage()*. Denne funksjonen kaller *getMessage()* som gjennom *getMessageUDP()* kaller socket-operasjonen *recvfrom()*. Ettersom noen av HADB-prosessene ikke er flertrådede kan hen-

ting av meldinger gjøres ved blokkering eller polling. Når en prosess ikke har noen andre viktige oppgaver, vil den gå i blokkerende venting. Dersom det ikke kommer en melding innen 20 ms, vil ventingen avbrytes og prosessen vil utføre ventende, lavere prioriterte oppgaver.

Ettersom HADB benytter upålitelig kommunikasjon gjennom UDP, inneholder meldingslaget logikk for eliminering av duplikate meldinger og retransmisjon av tapte meldinger.

HADB benytter et egendefinert meldingsformat som blant annet inneholder sekvensnummer, avsender og kvittering for tidligere meldinger. Dersom en melding er delt opp i flere deler, finnes også et fragmentnummer slik at meldingen kan settes sammen riktig hos mottaker.

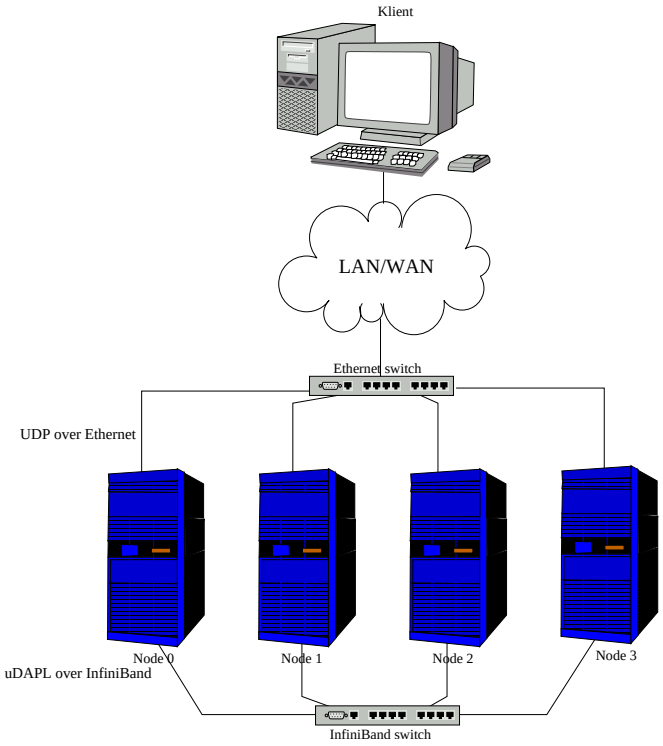
10.3.3 Meldingsheap

Meldingsheapen er et minneområde som brukes av både meldingslag og dialoglag. Innkommende meldinger legges i meldingsheapen ved mottak og slettes når systemet ikke lenger har behov for dem. Utgående meldinger opprettes i meldingsheapen, og en melding slettes når mottaker har bekreftet mottak av meldingen.

10.4 Endring av kommunikasjonssystemet

Dagens HADB kommuniserer, som tidligere nevnt, over UDP. Dette gjelder både kommunikasjon internt mellom noder og mellom en node og en klientapplikasjon. Ettersom det ikke kan forventes at klientapplikasjoner kjører på samme lokalnettverk som databasen, vil det være ønskelig at UDP-støtten beholdes for bruk mot disse. Samtidig skal nodene internt kommunisere over InfiniBand ved bruk av uDAPL. Dette er vist i figur 10.2.

Ettersom kommunikasjonssystemet i HADB er stort og komplisert (flere tusen kodelinjer i C), ble det grunnet stramt tidsbudsjett valgt å endre så lite av kommunikasjonssystemet som mulig. Bruk av en upålitelig protokoll som UDP medfører, som tidligere nevnt, at det finnes logikk for retransmisjon og dupliketeliminering. Ettersom uDAPL over InfiniBand bare tilbyr pålitelig kommunikasjon, kunne dette vært fjernet for intern kommunikasjon mellom nodene. Mot klientapplikasjoner må derimot denne logikken beholdes ettersom kommunikasjonen her fortsatt vil gå over UDP.



Figur 10.2: Kommunikasjon mellom noder og databasesystem

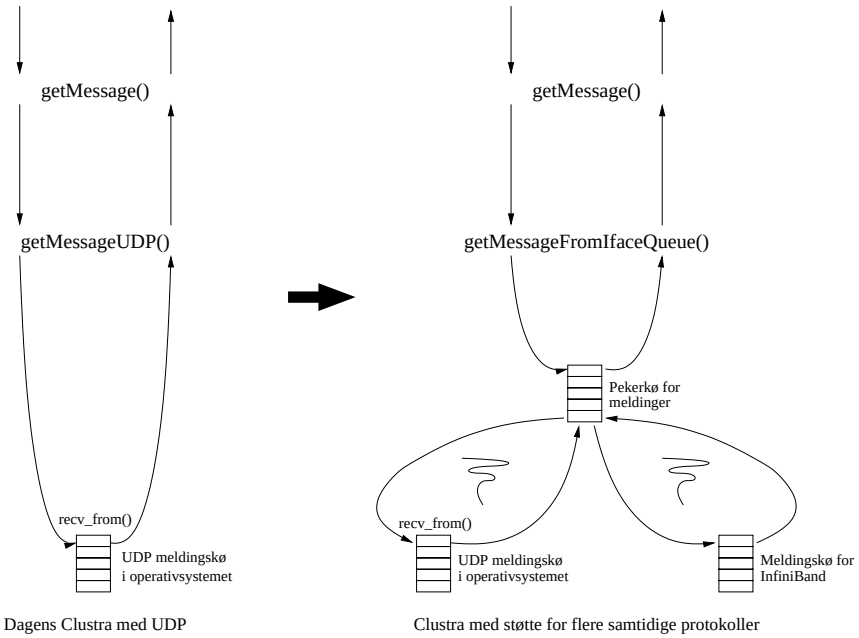
10.4.1 Støtte for flere protokoller

For at nodene skulle kunne kommunisere både over UDP og uDAPL, ble det gjort en endring i kallstakken. I dagens HADB kalles som nevnt *getMessage()* som gjennom *getMessageUDP()* henter en melding fra socketen. Endringen som ble gjort innebar at det her ble laget en ny funksjon kalt *getMessageFromIfaceQueue()*. Denne metoden hentet meldinger fra en mutex-beskyttet FIFO-kø kalt *ifaceQueue*, som inneholdt pekere til meldinger. I tillegg ble det innført to tråder. Bruken av kø og lesing av meldinger ved bruk av tråder er illustrert i figur 10.3.

Den ene tråden utførte blokkerende lesing av UDP-meldinger ved bruk av *getMessageUDP()*. Meldingene ble kopiert over i meldingsheapen og en peker til en melding ble lagt inn i pekerkøen.

Den andre tråden leste meldinger fra de forskjellige uDAPL-forbindelsene og la på tilsvarende måte disse inn i pekerkøen. Dette ble utført ved å lese hendelser fra mottakskøene og lese ut meldinger fra bufferne hendelsene pekte på. Samtidig var også denne tråden ansvarlig for å svare på innkommende tilkobling- og frakoblingshendelser fra andre prosesser. Tråden måtte derfor også behandle hendelser fra de tilkoblingsrelaterte hendelseskøene.

For å hindre utsulting av forbindelser, utførte tråden strukturert polling av



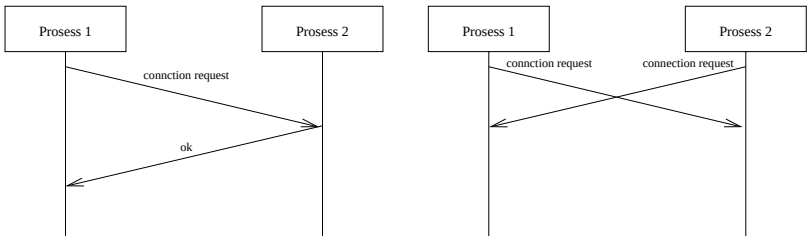
Figur 10.3: Bruk av pekerkø og mottak av meldinger over flere protokoller

hendelseskøene. For å hindre at tråden brukte mye CPU på polling i de tilfeller der det ikke var hendelser i køene, ble det også gjort bruk av CNO for hendelseskøene. Dersom det ikke var hendelser på noen køer, ble tråden stående å vente på CNO. Ved mottak av en hendelse, ble tråden vekket av CNO, og tråden startet utlesing av nye meldinger.

10.4.2 Oppretting av forbindelser

Ettersom UDP er forbindelsesløs mens uDAPL bare tilbyr forbindelsesorientert kommunikasjon, oppstod det enkelte problemer. Ved bruk av UDP mottas alle meldinger på en socket som lytter på en bestemt port. Meldingssystemet bruker portnummer, IP-adresse og informasjon fra konfigurasjonsfila for å bestemme hvilken prosess den prater med. Ved bruk av forbindelsesorientert kommunikasjon kreves én tilkobling for hver prosess en prosess vil kommunisere med. Endepunktet for en tilkobling mot en prosess vil dermed ikke ha samme *Connection Qualifier* for alle prosesser.

Det ovenfornevnte problemet ble løst ved to grep. De forskjellige prosessene oppretter et Public Service Point som har samme Connection Qualifier som prosessens portnummer. Ettersom en prosess leser portnummer og IP-adresse fra konfigurasjonsfila, vil den dermed enkelt kunne hvordan den skal foreta uDAPL-tilkoblingen. Samtidig er det mulig å inkludere ekstra informasjon i en tilkoblingsforespørsel i uDAPL. Dette ble utnyttet ved at en prosess pig-



Figur 10.4: Forenklet illustrasjon av etablering av tilkobling mellom to prosesser

gybacket egen IP-adresse og portnummer for UDP på forespørselen. Dermed kunne mottakende prosess enkelt identifisere hvilken prosess som sendte tilkoblingsforespørselen.

Forbindelsesorientert kommunikasjon medfører også et annet problem. Når en prosess skal koble seg til en annen, vil den sende en tilkoblingsforespørsel, og mottaker vil svare på denne (se venstre del av figur 10.4). Dersom to prosesser forsøker å koble seg til hverandre samtidig, må en av disse oppkoblingsforsøkene stanses (se høyre del av figur 10.4) slik at det ikke opprettes flere forbindelser enn nødvendig. Dette problemet ble løst ved at det alltid var den av to kommuniserende prosesser som hadde høyest portnummer som sendte tilkoblingsforespørselen. Før forbindelsen ble opprettet, ble UDP brukt mellom prosessene.

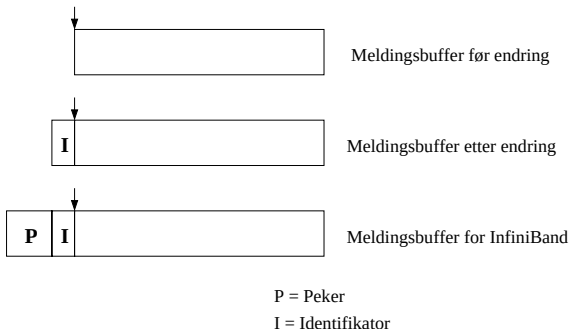
10.4.3 Ressursforvalting og buffere

Hver tilkobling ble gitt to sett buffere i ringstruktur på samme måte som i ytelsestesten (se kapittel 9.1.3): ett for mottak av meldinger og ett for sending. Disse buffersettene, samt IP-adresse og pekere til blant annet endpoint og hendelseskøer, ble lagt i en egen datastruktur kalt ibPort. De forskjellige ibPortene ble organisert i en hashtabell, og portnummer ble brukt som nøkkel til hashfunksjonen. Bufferne var av fast størrelse.

Ved sending av melding, ble det gjort oppslag i hashtabellen for å finne riktig ibPort. Ettersom bestemte interne strukturer brukes i HADB ved sending av melding, ble også ibPorten knyttet opp mot disse for å spare hashoppslaget når dette var mulig.

Ved mottak av meldinger, kunne riktig ibPort nås via pekere fra buffer. En peker til buffer ble returnert som en del av hendelsen som tilsa at data var mottatt. Gjenfinning av riktig ibPort ved mottak av melding var nødvendig både for å kunne poste bufferen på ny og for å kunne modifisere avsenderadresse i meldingene slik at bruk av InfiniBand ble skjult for høyere lag i systemet.

For allokering og frigjøring av minneområder i meldingsheapen i HADB, be-



Figur 10.5: Allokering av minneområder og buffere

nyttes tre kall: *allocSpace()*, *reallocSpace()* og *freeSpace*. Kallet *allocSpace()* tar størrelse som parameter og allokerer et minneområdet ved bruk av C-kallet *malloc()*. Kallet tar størrelse som parameter, allokerer et minneområde av ønsket størrelse og returnerer adressen til første byte i området. Kallet *reallocSpace()* endrer størrelsen på et tidligere allokert område ved bruk av C-kallet *realloc()*, og funksjonen tar adresse til gammelt område og ny størrelse som parametere. Dersom området skal gjøres større og det ikke er tilstrekkelig ledig minne i forlengelsen av det allokerte området, allokeres det et nytt område i riktig størrelse et annet sted. Alle data i det gamle området kopieres over i det nye og det gamle frigjøres. Kallet returnerer adressen til første byte i det nye området. For frigjøring av minneområder brukes kallet *freeSpace()*. Dette kallet tar et en peker til et minneområde som parameter og frigjør dette ved bruk av C-kallet *free()*.

InfiniBand-bufferne ble også allokert ved hjelp av *malloc()*. Samtidig ble disse registrert hos InfiniBand-hardwaren ved kallet *dat_lmr_create()* slik at relativ adresse kunne oversettes til fysisk. Dette er en CPU-krevende operasjon, og den bør derfor bare utføres ved oppretting av buffersettene.

For å unngå å kopiere mottatte meldinger, var det ønskelig å kunne la en melding ligge i InfiniBand-bufferet helt til høyere lag var ferdig med meldingen. Men ved frigjøring av en melding ble som nevnt *free()* benyttet. Det var ikke ønskelig å utføre dette kallet på et InfiniBand-buffer ettersom dette ville medført at bufferen måtte allokeres og registreres på nytt for å kunne benyttes til nye meldingsmottak.

Det ble utført tre endinger i meldingsheapen for å unngå dette. I *allocSpace()* ble det allokert en byte ekstra i starten av minneområdet. Denne første byten ble brukt for å fortelle at minneområdet tilhørte meldingsheapen. Samtidig returnerte *allocSpace()* adressen til byte nummer to. Denne ekstra byten ble dermed skjult, og de fleste andre funksjoner kunne benytte meldingsområdet uten endring. På tilsvarende måte ble det allokert en byte ekstra ved allokering av minneområde til InfiniBand-bufferne. I tillegg ble det også allokert plass til

en peker. Denne pekeren ble satt til å peke på buffer-structen minneområdet tilhørte, slik at tilhørende ressurser enkelt kunne gjenfinnes ved reposting av bufferen. Figur 10.5 viser hvordan minneområdene ble seendes ut etter endringen. Pilen viser adressen som ble gitt til andre funksjoner i meldingslaget for å skjule endringen.

Ved kall til *reallocSpace()*, ble det sjekket om minneområdet som skulle endres var en InfiniBand-buffer. Ettersom *reallocSpace()* fikk adressen til byte nummer to som parameter fra andre funksjoner, ble sjekken utført ved å undersøke byten som lå like før pekeren som ble angitt som parameter. Dersom minneområdet var en del av meldingsheapen, ble *realloc()* utført som tidligere. Dersom området var en InfiniBand-buffer, ble det sjekket om den nye størrelsen var større enn den faste bufferstørrelsen. Var den det, ble det opprettet et nytt område på meldingsheapen. Meldingen ble kopiert over og InfiniBand-bufferen ble gjort tilgjengelig for mottak av nye meldinger. Dersom den nye størrelsen var mindre enn den faste buffer-størrelsen, var minneområdet allerede stort nok og samme peker ble returnert.

Ved frigjøring av minneområder ved *freeSpace()*, ble det tilsvarende sjekket om minneområdet tilhørte meldingsheapen eller var en InfiniBand-buffer. Dersom minneområdet tilhørte meldingsheapen, ble *free()* benyttet som før endringen. Var det en InfiniBand-buffer, ble bufferen gjort tilgjengelig for mottak av nye meldinger.

10.4.4 Begrensninger ved sending

I meldingsheapen til HADB kan det allokeres meldinger i vilkårlig størrelse, og disse kan senere endre størrelse. Samtidig er det ikke alltid åpenbart idet minneområdet allokeres hvilken prosess som skal motta meldinger fra området.

Ettersom uDAPL-bufferne må forhåndsallokeres for en forbindelse, må det være kjent hvilken prosess som skal motta data idet bufferet velges ut. Videre krever semantikken for uDAPLs meldingssending at data i en meldingsbuffer ikke endres mellom kall til send og mottak av hendelse som tilsier at sendingen er utført.

Fjerning av kopiering ved sending ville krevd store endringer i kommunikasjonssystemet, og det ble av tidshensyn valgt å bare fjerne kopiering av meldinger ved mottak. Ved sending kopieres meldinger fremdeles over i nye buffere ved sending slik som ved UDP. Men bruk av innebygd DMA-kontroller og protokollstakk som i stor grad utføres i hardware antas likevel å gi en ytelsesøkning.

Dersom en melding er for stor for en buffer ved sending over UDP, deler HADB meldingen i mindre biter og sender hver bit for seg. Bitene kopieres da over i egne buffere og gis ny header innholdende fragmentnummer. Ved bruk av

uDAPL ble det her gjort et grep slik at meldingene ble kopiert direkte over i uDAPL-buffere når de skulle gis egne headere. Dermed unngås en minne-til-minne-kopiering når meldingsstørrelsen er større enn maksimal UDP-størrelse for systemet (32768 byte).

10.4.5 Justeringsparametre for løsning

I den ferdige løsningen finnes det noen parametere som kan styre ytelse og ressursforbruk for uDAPL. Bufferantall og bufferstørrelse kan, som vist i kapittel 9.3, være avgjørende for ytelsen til det ferdige systemet. Samtidig bindes minnet brukt til buffere og kan ikke brukes til andre ting.

Ved for lavt bufferantall vil systemet vente på ledige buffere eller eventuelt kaste meldinger det ikke får sent. Ved for lav bufferstørrelse vil mange meldinger bli delt i flere mindre meldinger, noe som impliserer noe ekstraarbeid.

Ved for høyt bufferantall vil ikke systemet utnytte alle bufferne. Noen buffere vil være ute av bruk, og disse vil bare legge beslag på minne. Tilsvarende vil for store buffere ikke kunne utnyttes av systemet.

Antall buffere og størrelsen på disse må derfor vurderes mot tilgjengelig minne og type last.

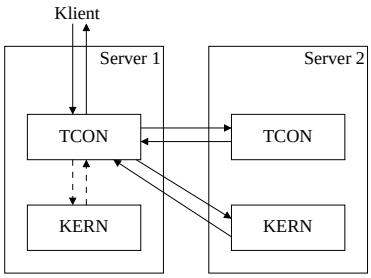
10.5 Testbeskrivelse og utførelse

Testene ble utført på samme servere som ytelsestesten (se kapittel 9.2). Kildekoden til HADB ble sjekket ut av CVS 21. januar kl. 09:08. Kildekoden ble kompilert med kompilatorer fra Sun ONE Studio 8, og flagg ble satt for fjerning av debug-kode. Databasen ble kompilert i en UDP- og en uDAPL-utgave for utføring av testene. Versjon 1.1 av uDAPL ble benyttet. Bufferantall var satt til 40 for sending og 60 for mottak, og bufferstørrelsen var satt til 8192 byte. Databasesystemet hadde to noder (en node på hver server).

Testprogrammene beskrevet videre i delkapitlet, er laget av utviklerne av HADB og brukes ved benchmarking og testing av databasesystemet. Programmene er skrevet i C eller Java.

Ved 1tup-read-, 4tup-run- og TPC-B-testene ble default-verdier i konfigurasjonsfila brukt for innstilling av databasesystemet. Ved BLOB-testene ble innstillingene endret noe, for å kompensere for økning i tabellstørrelse (nbuffs ble økt fra 4096 til 40960 og logsize fra 16777216 til 167772160). Ettersom det, spesielt i 10 KB BLOB-testene, ble forventet utveksling av flere store meldinger, ble også bufferstørrelsen i uDAPL-koden økt fra 8192 byte til 32768 byte.

Testene ble ikke utført på IP over InfiniBand grunnet dårlige resultater fra



Figur 10.6: Meldinger ved en 1tup-read (basert på figur 51 i [MS00])

ytelsestesten.

10.5.1 1tup-read

1tup-read er beskrevet i [MS00], og er et klientprogram som kobler seg opp databasen og foretar lesetransaksjoner. Hver lesetransaksjon leser ett tilfeldig tuppel fra databasen, og klienten starter ikke en ny transaksjon før den forrige er utført.

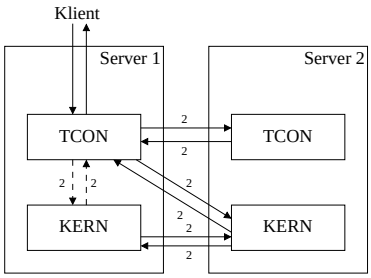
Figur 10.6 viser meldingsflyten ved bruk av klienten. De stiplede pilene illustrerer meldinger som går internt i prosessen, mens de heltrukne illustrerer meldinger som sendes mellom prosesser. Ved 1tup-read sendes det to meldinger dersom det etterspurte tuppelet ligger på serveren klienten er koblet til, eller seks dersom tuppelet ligger på server nummer to. Ettersom klienten kjører på den ene serveren, er det bare meldingene mellom de to serverne som overføres over nettverket.

Det ble utført tre forskjellige tester med denne klienten. I den første testen var klienten bare koblet til en av nodene, og den ble også kjørt på samme server som noden den var koblet til. I test nummer to ble det kjørt en klient på hver server, og hver klient var koblet til samme server som de kjørte på. I den tredje testen ble det kjørt to klienter per server, og disse var koblet til samme server som de kjørte på.

I hver test ble klientene satt til å kjøre transaksjoner i 600 sekunder, og det var ingen ventetid (*thinktime*) fra en transaksjon var avsluttet til den neste ble startet. Klienten utførte forespørsler mot en tabell med 50 000 innslag.

10.5.2 4tup-run

4tup-run er beskrevet i [MS00], og er på samme måte som 1tup-read en klient som kobler seg til en av serverne i databasen. Klienten sender oppdateringstransaksjoner der hver transaksjon endrer fire tupler, og den venter på resultat



Figur 10.7: Meldinger ved en 4tup-run (basert på figur 52 i [MS00])

fra en transaksjon før den starter den neste.

Oppdateringstransaksjonene krever blant annet oversending av ny verdi og oppdatering av logg, og det sendes dermed flere meldinger mellom prosessene i databasen enn ved 1tup-read. Figur 10.7 illustrerer antall meldinger som overføres. Dersom tupplene ligger på serveren noden er koblet til, sendes 12 meldinger, mens det sendes 14 meldinger dersom tuplene ligger på node nummer to.

På samme måte som for 1tup-read, ble det utført tre tester ved bruk av dette programmet. I den første testen kjørte en klient på den ene noden, mens det i test nummer to og tre ble kjørt henholdsvis én og to klienter på hver node.

I hver test kjørte klientene transaksjoner mot databasen i 600 sekunder, og det var ingen ventetid (*thinktime*) fra en transaksjon var ferdig til den neste ble startet. Klientene utførte forespørsler mot en tabell med 50 000 innslag.

10.5.3 TPC-B

TPC-B [tpc01] er transaksjonsbasert last for ytelsestesting av databaser. Scenarioet bak er en bank, og transaksjonene symboliserer kunder som setter inn eller tar ut penger fra konto. En innføring i TPC-B finnes på [GC98]

TPC-B-testene ble utført ved bruk av en Java-basert testklient som koblet seg til databasen via JDBC. Innstillingene for testen er vist i tabell 10.1. Testen ble kjørt to ganger. Den første testen kjørte i 600 sekunder og hadde to tråder som koblet seg til databasen og kjørte transaksjoner. Den andre testen kjørte også i 600 sekunder, men denne testen hadde fire tråder. I begge tilfeller kjørte klienten på den ene serveren, men noen tråder koblet seg til den andre serveren.

Parameter	Verdi
Branches	64
Tellers	640
Accounts	64 000

Tabell 10.1: Innstillinger for TPC-B test

Størrelse	Innsetting	Lesing	Oppdatering	Sletting
1 KB	10 000 operasjoner	120 sekunder	120 sekunder	10 000 operasjoner
10 KB	5000 operasjoner	120 sekunder	120 sekunder	5000 operasjoner

Tabell 10.2: Parametre ved BLOB-test

10.5.4 BLOB

En BLOB, eller *Binary Large Object*, er som navnet tilsier et binært objekt og kan for eksempel være et bilde eller en tekstfil. BLOB-testen ble utført ved bruk av en Java-basert klient som benytter JDBC. Testen ble utført med to forskjellige BLOB-størrelser: 1 KB og 10 KB. I hver test ble det utført fire deltester: innsetting, lesing, oppdatering og sletting. Tabell 10.2 viser hvilke innstillinger som ble gjort for å styre varigheten av de forskjellige deltestene.

10.6 Resultater

10.6.1 Resultatenes gyldighet

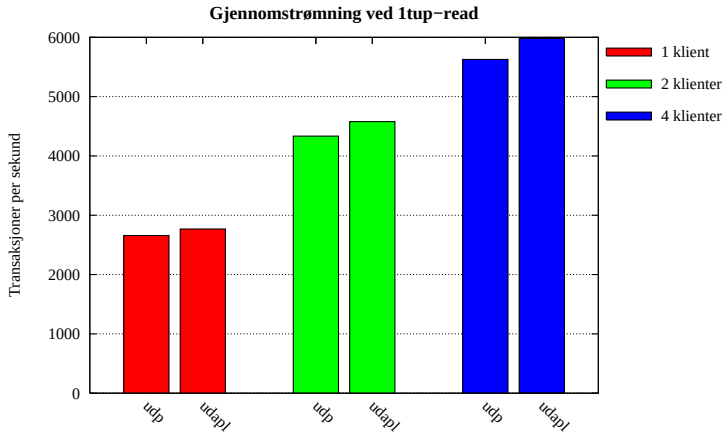
Resultatene er bare gyldige for versjon Sun HADB sjekket ut av CVS 21. januar kl. 09:08 og for oppgitt hardware. Ettersom verken database eller tester er konfigurert for optimal ytelse, må ikke resultatene ses på som absolutte mål for ytelsen Sun HADB kan oppnå. Testene er kun ment å vise hvordan forskjell i kommunikasjonsform kan påvirke ytelsen.

10.6.2 1tup-read

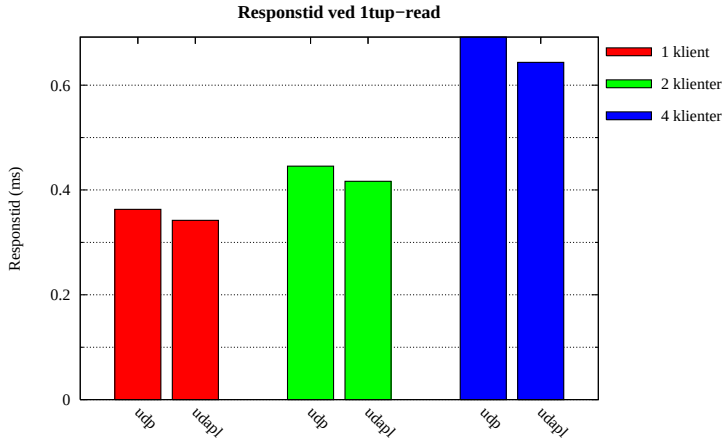
Resultatene fra 1tup-read-testene er gjengitt i tabell 10.3, og forskjeller i gjennomstrømning og gjennomsnittlig responstid er illustrert i henholdsvis graf 10.1 og graf 10.2. UDP-versjonen har den laveste minimale responstiden, men har også høyest maksimal responstid. uDAPL-versjonen har 6-7% lavere gjennomsnittlig responstid og 4-6% høyere gjennomstrømning.

	Antall klienter	Utførte transaksjoner	Gjennom-strømning	Gjennomsnittlig responstid	Minimal responstid	Maksimal responstid
UDP	1	1 594 824	2 658,0 tps	0,36 ms	0,20 ms	38,10 ms
	2	2 600 584	4 334,3 tps	0,45 ms	0,20 ms	30,65 ms
	4	3 376 106	5 626,8 tps	0,69 ms	0,19 ms	31,73 ms
uDAPL	1	1 661 047	2 768,4 tps	0,34 ms	0,22 ms	22,53 ms
	2	2 747 160	4 578,6 tps	0,42 ms	0,22 ms	26,12 ms
	4	3 588 548	5 980,9 tps	0,64 ms	0,21 ms	19,86 ms

Tabell 10.3: Resultater fra 1tup-read



Graf 10.1: Gjennomstrømning ved kjøring av 1tup-read



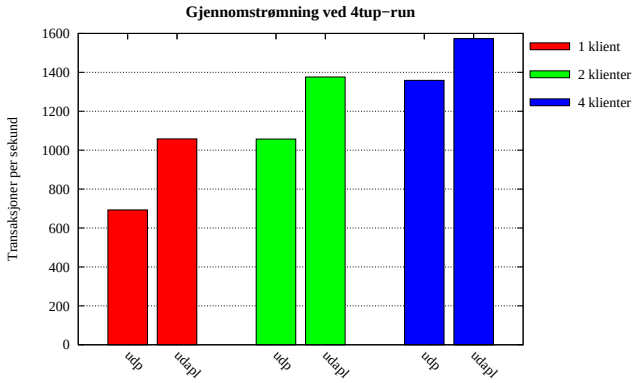
Graf 10.2: Gjennomsnittlig responstid ved kjøring av 1tup-read

10.6.3 4tup-run

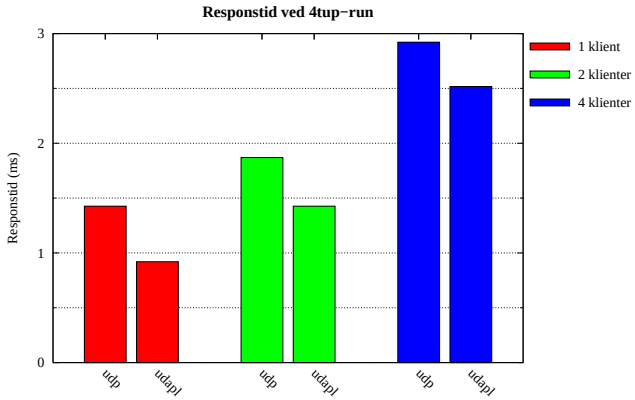
Resultatene fra 4tup-run-testen er gjengitt i tabell 10.4, og forskjellene i gjennomstrømning og gjennomsnittlig responstid er illustrert i henholdsvis tabell graf 10.3 og graf 10.4. I denne testen har uDAPL-versjonen både lavere minimal, gjennomsnittlig og maksimal responstid. Den har 15-53% høyere gjennomstrømning og 14-35% lavere gjennomsnittlig responstid.

	Antall klienter	Utførte transaksjoner	Gjennom-strømning	Gjennomsnittlig responstid	Minimal responstid	Maksimal responstid
UDP	1	415 719	692,9 tps	1,43 ms	0,86 ms	500,57 ms
	2	634 401	1 057,3 tps	1,87 ms	0,92 ms	467,95 ms
	4	815 236	1 358,8 tps	2,92 ms	1,13 ms	487,95 ms
uDAPL	1	634 750	1 057,9 tps	0,92 ms	0,69 ms	21,84 ms
	2	825 621	1 376,0 tps	1,43 ms	0,73 ms	13,18 ms
	4	943 961	1 573,2 tps	2,52 ms	0,76 ms	91,16 ms

Tabell 10.4: Resultater fra 4tup-run



Graf 10.3: Gjennomstrømning ved kjøring av 4tup-run



Graf 10.4: Gjennomsnittlig responstid ved kjøring av 4tup-tun

10.6.4 TPC-B

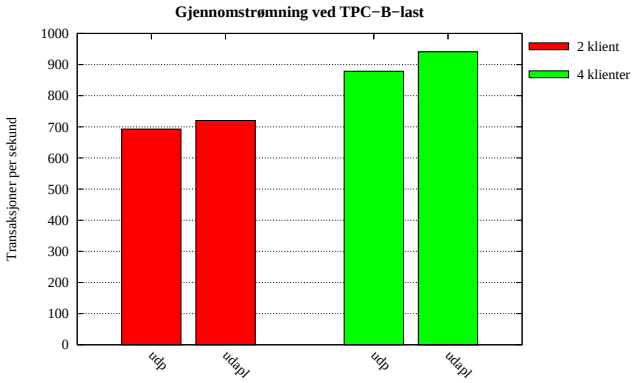
Resultatene fra TPC-B-testen er gjengitt i tabell 10.5, og forskjeller i gjennomstrømning og gjennomsnittlig responstid er gjengitt i henholdsvis graf 10.5 og 10.6. Resultatene viser at uDAPL-versjonen har 3-7% høyere gjennomstrømning og 4-6% lavere responstid.

	Antall klienter	Utførte transaksjoner	Gjennom-strømning	Gjennomsnittlig responstid
UDP	1	415 884	692,9 tps	2,88 ms
	2	527 951	878,6 tps	4,54 ms
uDAPL	1	432 474	720,3 tps	2,77 ms
	2	565 469	941,0 tps	4,24 ms

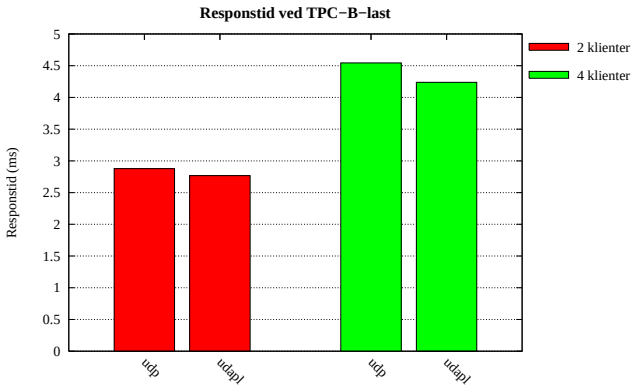
Tabell 10.5: Resultater fra TPC-B-testen

10.6.5 BLOB

Tabell 10.6 gjengir resultatene fra 1KB BLOB-testen, og graf 10.7 og graf 10.8 illustrerer forskjeller i henholdsvis gjennomstrømning og responstid for testen.



Graf 10.5: Gjennomstrømning ved TPC-B-last



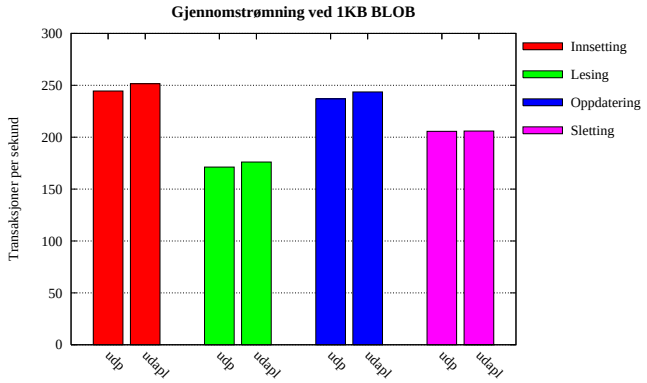
Graf 10.6: Gjennomsnittlig responstid ved TPC-B-last

UDP- og uDAPL-versjonene har relativ lik ytelse ved sletting, mens uDAPL-versjonen er noe bedre ved innsetting, lesing og oppdatering. uDAPL har ca. 3% høyere gjennomstrømning ved de tre sistnevnte testene.

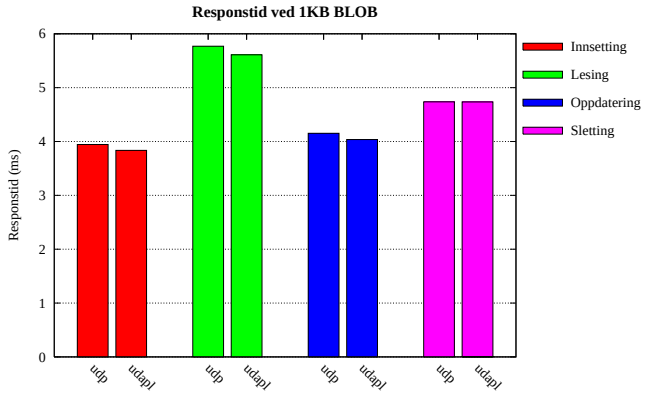
På tilsvarende måte som for 1 KB BLOB, gjengir tabell 10.7 resultatene fra 10 KB BLOB-testen, mens graf 10.9 og graf 10.10 illustrerer henholdsvis forskjeller i gjennomstrømning og gjennomsnittlig responstid. Også her har uDAPL og UDP relativt lik ytelse ved sletting, mens uDAPL har 5-11% høyere gjennomstrømning ved de andre deltestene.

	Deltest	**Utførte transaksjoner**	**Gjennom-strømning**	**Gjennomsnittlig responstid**
UDP	Innsetting	9 874	244,5 tps	3,94 ms
	Lesing	20 544	171,2 tps	5,77 ms
	Oppdatering	28 431	236,9 tps	4,15 ms
	Sletting	9 863	205,7 tps	4,73 ms
uDAPL	Innsetting	9 873	251,6 tps	3,83 ms
	Lesing	21 131	176,1 tps	5,61 ms
	Oppdatering	29 235	243,6 tps	4,03 ms
	Sletting	9 860	205,9 tps	4,74 ms

Tabell 10.6: Resultater fra 1KB BLOB-test



Graf 10.7: Gjennomstrømning ved 1KB BLOB-transaksjoner



Graf 10.8: Gjennomsnittlig responstid ved 1KB BLOB-transaksjoner

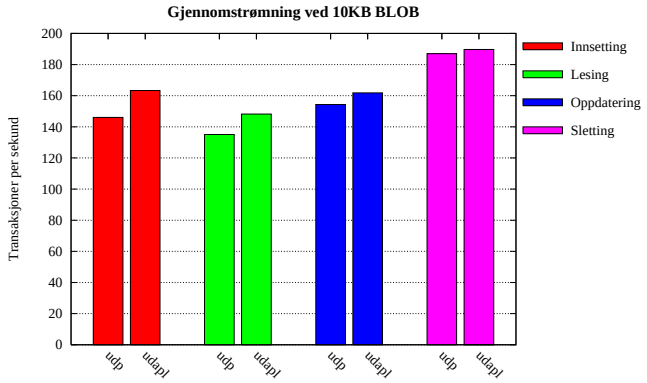
	Deltest	Utførte transaksjoner	Gjennom-strømning	Gjennomsnittlig responstid
UDP	Innsetting	4 978	146,1 tps	6,72 ms
	Lesing	16 209	135,1 tps	7,33 ms
	Oppdatering	18 521	154,3 tps	6,41 ms
	Sletting	4 963	187,0 tps	5,24 ms
uDAPL	Insetting	4 977	163,3 tps	6,00 ms
	Lesing	17 784	148,2 tps	6,68 ms
	Oppdatering	19 417	161,8 tps	6,11 ms
	Sletting	4 966	189,7 tps	5,17 ms

Tabell 10.7: Resultater fra 10KB BLOB-test

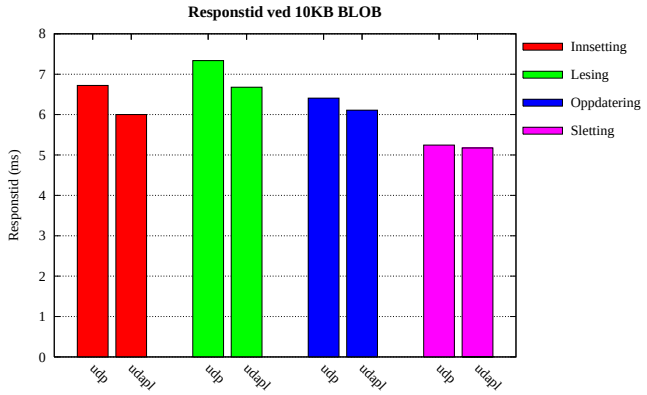
10.7 Diskusjon

Resultatene viser at uDAPL-versjonen i 1tup-read- og 4tup-run-testene yter merkbart bedre enn UDP-versjonen. Den største forskjellen mellom de to versjonene finnes i resultatene fra 4tup-run testene. Grunnen til at forskjellen er større ved 4tup-run enn ved 1tup-read, skyldes trolig at det sendes mye flere meldinger ved 4tup-run.

Et resultat som er verdt å merke seg, er at UDP-versjonen har lavere minimal responstid ved 1tup-read. Dette kan skyldes det ekstra steget innom pekerkøen



Graf 10.9: Gjennomstrømning ved 10KB BLOB-transaksjoner



Graf 10.10: Gjennomsnittlig responstid ved 10KB BLOB-transaksjoner

som meldingene tar. Ved 4tup-run har UDP-versjonen langt høyere maksimal responstid, noe som tyder på at uDAPL-versjonen gir en mer stabil responstid for disse transaksjonene.

TPC-B- og BLOB-testene viser ikke samme forskjell mellom UDP- og uDAPL-versjonen som 1tup-read og 4tup-run, og fra kjøring til kjøring var det variasjoner i hvor stor ytelseforbedringen var. Det er noe usikkert hva som er grunnen til dette. En mulig grunn kan være forskjeller i testutførelse. 1tup-read og 4tup-run kobler seg direkte til clu_trans_srv-prosessen og sender transaksjonene til denne. TPC-B og BLOB-testen kobler seg derimot til SQL-serveren clu_sql_srv. Denne prosessen starter en egen SQLSUB-prosess som tar seg av de videre forespørslene fra klientapplikasjonen og videreformidler transaksjoner til clu_trans_srv-prosessen for utførelse. Klientapplikasjonen velger også tilfeldig hvilken av de to nodene den vil koble seg til. Så selv om klientapplikasjonen kjører på samme server som node null, kan den koble seg til SQL-serveren på den node en og motsatt. Ved BLOB-operasjoner varierer det fra deltest til deltest hvilken node klientapplikasjonen kobler seg til, og det kan her være forskjeller i hvilken node testapplikasjonen har koblet seg til i UDP og uDAPL-versjonen. Dersom klientapplikasjonen kobler seg til SQL-serveren på samme node, vil

meldingene aldri overføres over nettverket. Dette kan påvirke ytelsen ettersom det ved bruk av Ethernet er mulig å oppnå langt høyere ytelse mellom to prosesser på samme maskin enn det er mulig å oppnå dersom prosessene finnes på separate maskiner. Ved bruk av for eksempel 100 Mbit Ethernet er det mulig å oppnå en langt høyere overføringsrate enn 100 Mbit/s. Dette skyldes at pakkene aldri overføres over nettverket, og ytelsen påvirkes dermed bare av kjøretid for traversering av IP-stakk og overføringsrate ved minne-til-minne-kopiering.

Det må også nevnes at alle testene påvirkes av at kopiering ikke er fjernet ved sending av meldinger. Tallene fra ytelsestesten i kapittel 9 viser at potensialet som ligger uDAPL og InfiniBand vil kunne medføre høyere ytelse i en fullstendig implementasjon enn det som er vist her. Temporære tester utført under implementasjonen viste blant annet at fjerning av kopiering ved mottak, hadde stor innvirkning på ytelsen. Med kopiering ved mottak var det nesten ingen ytelsesforbedring ved 4tup-run. Selv om også andre forbedringer er utført etter at kopiering ved mottak ble fjernet, antas det at fjerning av kopiering ved sending vil gi en merkbar ytelsesforbedring.

Kapittel 11

Konklusjon

Et parallelt shared-nothing databasesystem har høy meldingsutveksling mellom nodene og er derfor avhengig av effektiv kommunikasjon. Bruk av standard IP-protokoll medfører stor belastning på CPU for kopiering av pakker og traversering av IP-protokoll med de operasjoner det innebærer.

Gjennom oppgaven er det presentert forskjellige teknologier kommunikasjon både internt på hovedkortet og mellom nodene. InfiniBand og uDAPL er blitt introdusert som et alternativ til IP-protokollen internt mellom nodene i databasen. Ved bruk av uDAPL over InfiniBand vil ikke meldingene gå innom operativsystemet, men kan sendes fra direkte fra applikasjon til applikasjon. På den måten unngås minne-til-minne-kopiering.

Ytelsestesten viste at bruk av InfiniBand og uDAPL kan redusere CPU-forbruket og øke overføringsraten samtidig. Testen viste at uDAPL spesielt er overlegen ved høy meldingsstørrelse.

Testimplementasjonen i Sun HADB viste at bruk av uDAPL og InfiniBand kan øke gjennomstrømning og senke responstid for flere typer transaksjoner. Ettersom implementasjonen måtte begrenses grunnet tidshensyn, ble resultatene noe varierte. Samlet tilsier likevel resultatene fra ytelsestesten og implementasjonstesten at det i en fullstendig implementasjon er mye å hente i bruk av uDAPL og InfiniBand.

Kapittel 12

Videre arbeide

For å gi et fullstendig bilde, ville direkte sammenligning av flere forskjellige alternativer vært ønskelig. Vurdering av for eksempel zero-copy-implementasjoner, TCP-offload-engines og InfiniBand utført på samme server, ville gitt et bedre bilde av hvordan de forskjellige løsningene oppfører seg i forhold til hverandre.

Nærmere undersøkelser rundt den noe lavere ytelsesøkningen i BLOB og TPC-B testene kunne også være av interesse for å kartlegge de faktorer som spiller inn. Sammenligning av reperasjonstid ved nodekrasj for UDP- og uDAPL-løsningene kunne også gitt et bilde av hvordan uDAPL kunne påvirket tilgjengeligheten direkte som en følge av økt kommunikasjonsytelse.

Det kunne også vært undersøkt om bruk av en tråd per tilkobling ville gitt bedre ytelse, ettersom dette ville medført langt færre pollinger av tomme hendelseskøer.

En fullstendig implementasjon med fjerning av kopiering ved sending i Sun HADB, ville også gitt et bedre bilde av hvordan IP-protokollen og uDAPL forholder seg til hverandre ytelsesmessig. For å redusere antall tomme buffere kunne en her benyttet bufferprofiler. Med bufferprofiler menes at forbindelser som selv ved høy last sender få små meldinger (for eksempel mellom nodeovervåkningsprosessene), ikke behøver like mange og like store buffere som mer kommunikasjonsintensive forbindelser.

Ordliste

API	*Application Programming Interface*
Bro	En chip som forlenger en buss eller samler flere segmenter av busser av samme eller forskjellige type
CRC	*Cyclic Redundancy Check* - checksumvariant
DMA	*Direct Memory Access* - overføring av data mellom enheter internt i en datamaskin uten involvering av prosessor
Full duplex	Sending og mottak kan utføres samtidig
Last	Arbeidsmengde
Latency	Responstid eller tid det tar å transportere en melding over en nettverksleder
Link	Linje mellom to naboenheter i et nettverk
Nedetid	Tid systemet er utilgjengelig
Node	Datamaskin i et parallelt eller distribuert databasesystem
OS-bypass	Sending av meldinger over nettverk fra applikasjon til applikasjon uten å gå via operativsystemet
Overhead	Ekstraarbeid utført for å støtte opp om andre funksjoner
Polle	Ikke-blokkerende venting
Poste buffer	Fortelle maskinvare at bufferen kan brukes til mottak av en melding eller at innholdet i bufferen skal sendes
Port	Tilkoblingspunkt eller kontakt på en nettverksadapter eller en switch
Router	Enhet som muliggjør sammenkobling av flere nettverk
RPC	*Remote Procedure Call*
Single point of failure	Enhet et system er avhengig av for å fungere
Switch	Knutepunkt for linker
Userspace	Brukerapplikasjons minneområde og context
Zero-copy-protokoll	En kommunikasjonsprotokoll som ikke foretar intern minne-til-minne-kopiering av meldingsdata

Bibliografi

[Amd01] HyperTransport Technology I/O Link. White paper, Advanced Micro Devices (AMD), 2001.

[Asp03] Advanced Switching for the PCI Express Architecture. White paper, Intel Corporation, 2003.

[BH01] Svein Erik Bratsberg and Rune Humborstad. Online scaling in a highly available database. 2001.

[Bha02] Ajay V. Bhatt. Creating a Third Generation I/O Interconnect. White paper, Intel Corporation, 2002.

[Bou03] Dan Bouvier. RapidIO: The Interconnect Architecture for High Performance Embedded Systems. Technology paper, Rapid IO Steering Committee, 2003.

[cis04] Ethernet technologies. Cisco Systems *http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ethernet.htm*, 2004.
Aksessert 27. januar 2004.

[Cpq99] PCI-X: An Evolution of the PCI Bus. Technology Brief, Compaq Computer Corporation, 1999.

[DAF] DAFS Collaborative. Direct Access File System. *http://www.dafscollaborative.org*.

[DAP] DAT Collaborative. Direct Access Programming Library. *http://www.datcollaborative.org*.

[dol96] The Doplhin SCI Interconnect. White paper, Dolphin Interconnect Solutions AS, 1996.

[dol04] PCI-SCI Adapter Card. Dolphin Interconnect Solutions AS *http://www.dolphinics.com/products/hardware/pci64.html*, 2004.
Aksessert 26. februar 2004.

[EM99] Daniel Elftmann and Jing Hua Ma. Implementing an 8B/10B Encoder/Decoder for Gigabit Ethernet. In *International IC Taipei '99 Conference Proceedings*, 1999.

[FIB] Fibre Channel Industry Association. Fibre Channel. *http://www.fibrechannel.org*.

[GC98] Jim Gray and Rick Cattell. The Benchmark Handbook. *http://www.benchmarkresources.com/handbook/tpcb-1.html*, 1998.
Aksessert 30. mai 2004.

[GCY99] Andrew Gallatin, Jeff Chase, and Ken Yocum. Trapeze/IP: TCP/IP at Near-Gigabit Speeds. In *Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference*, 1999.

[Gou02] Mathieu Goutelle. Queues in the Linux kernel, 2002.

[GR93] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and techniques*. Morgan Kaufmann Publishers, Inc., 1993.

[Gru03] Grant Grundler. DMA Hints on IA64/PARISC: Optimizing DMA performance for HP Chip sets, 2003. Hewlett Packard.

[GS91] Jim Gray and Daniel P. Siewiorek. High-availability computer systems, 1991.

[Hav02a] Yaron Haviv. Breaking Through The Bottleneck. Technical report, Voltaire Inc., 2002.

[Hav02b] Yaron Haviv. Making Room For The Application. Technical report, Voltaire Inc., 2002.

[Hva99] Svein-Olaf Hvasshovd. *Recovery in Parallel Database Systems*. Vieweg, 2 edition, 1999.

[hyp01] The HyperTransport Technology (HT) I/O Bus Architecture. White paper, API NetWorks, 2001.

[HyTBH95] Svein-Olaf Hvasshovd, Øystein Torbjørnsen, Svein Erik Bratsberg, and Per Holager. The ClustRa Telecom Database: High Availability, High Throughput, and Real-Time Response. In *VLDB'95, Proceedings of 21th International Conference on Very Large Data Baseses*, 1995.

[IBt] An InfiniBand Technology Overview. InfiniBand Trade Association
http://www.infinibandta.org/ibta/.

[IEE] IEEE 802.3 Ethernet. IEEE 802.3 Working Group
http://www.ieee802.org/3/.

[IPI] InfiniBand over IP.
http://www.ietf.org/html.charters/ipoib-charter.html.

[Kra02] Eric Krause. HyperTransport Technology and InfiniBand Architecture: The Complete Solution for High Bandwidth I/O Solution. White paper, Advanced Micro Devices (AMD), 2002.

[Mel00] InfiniBand and TCP in the Data Center. Technical report, Mellanox Technologies, 2000.

[Mel03] Understanding PCI Bus, PCI-Express and InfiniBand Architecture. Technical White Paper, Mellanox Technologies, 2003.

[Mic] Sun Microsystems. Privat samtale med Øystein Torbjørnsen.

[Mog03] Jeffrey C. Mogul. TCP Offloaf is a dumb idead whose time has come. Hotos ix paper, Hewlett-Packard Laboratories, 2003.

[MR] David Mayhew and Norm Rasmussen. PCI Express and Advanced Switching: Evolutionary Path to Revolutionary Architectures. White paper, ASI SIG.

[MS00] Christian Myksvoll and John Torger Skjelstad. Innføring av SCI i Clustra. Master's thesis, NTNU, 2000.

[myr04a] Myrinet Overview. Myricom
http://www.myri.com/myrinet/overview/index.html, 2004.
Aksessert 26. februar 2004.

[myr04b] Myrinet Software and Documentation. Myricom
http://www.myri.com/scs/index.html, 2004.
Aksessert 27. februar 2004.

[no-04] Intel-brikkesett støtter ikke AGP. Hardware.no
http://hardware.no/art.php?artikkelid=5347, 2004.
Aksessert 2. mars 2004.

[Pci] PCI Special Interest Group.
http://www.pcisig.com/.

[PSi] PCI-SIG Frequently Asked Questions. PCI Special Interest Group
http://www.pcisig.com/news_room/faqs.
Aksessert 09.februar 2004.

[qua04] QsNet High Performance Interconnect. Quadrics *http://doc.quadrics.com/Quadrics/QuadricsHome.nsf/DisplayPages/3A912204F260613680256DD9005122C7*, 2004. Aksessert 1. mars 2004.

[SDP] RDMA Consortium. Socket Direct Protocol. *http://www.rdmaconsortium.org/*.

[SOHL+96] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI: The Complete Reference*. The MIT Press, 1996. *http://www.netlib.org/utk/papers/mpi-book/mpi-book.ps*.

[SPGH01] Karl-André Skevik, Thomas Plagemann, Vera Goebel, and Pål Halvorsen. Evaluation of a zero-copy protocol implementation. Technical report, Universitetet i Oslo, 2001.

[SRP] ANSI T10 committee. SCSI RDMA Protocol. *http://www.t10.org*.

[Sun03] Horizontal Scaling Fabrics for Sun Fire V60x and V65x Servers: InfiniBand. Technical report, Sun Microsystems, 2003.

[Tan96] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, 3. edition, 1996. ISBN 0-13-349945-6.

[TO03] Jens Eirik Tengesdal and Erlend Oftedal. Refordeling av data ved skalering av parallelle databaser. Fordypningsprosjekt, NTNU, 2003.

[top03] Top 500 Supercomputer Sites 11/2003. TOP500 *http://www.top500.org/*, 2003. Aksessert 27. februar 2004.

[tpc01] TPC-B. *http://www.tpc.org/tpcb/default.asp*, 2001. Aksessert 30. mai 2004.

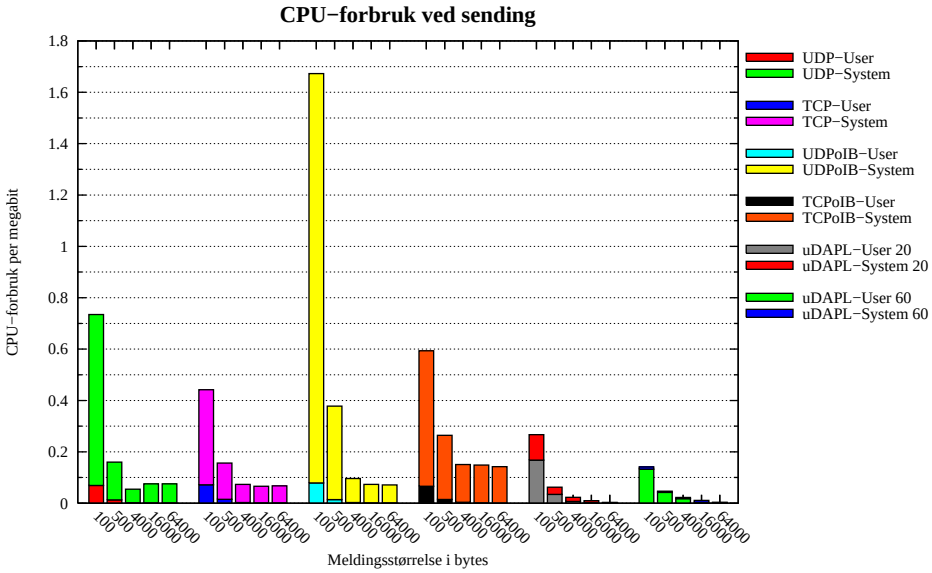
[Via97] Virtual Interface Architecture Specification, 1997. Compaq, Intel og Microsoft.

[YCM+02] Eric Yeh, Herman Chao, Venu Mannem, Joe Gervais, and Bradley Booth. Introduction to TCP/IP Offload Engine (TOE). Technical report, 10 Gigabit Ethernet Alliance, 2002.

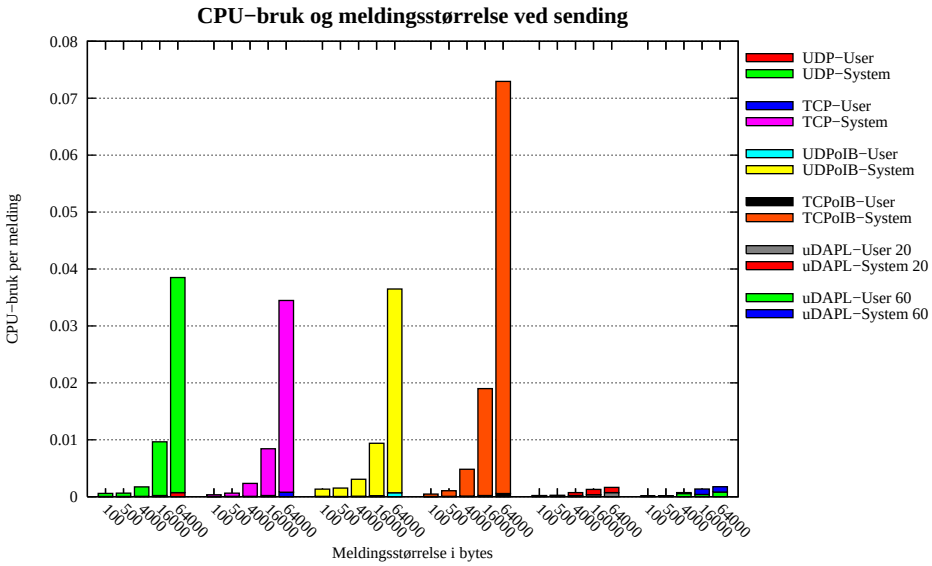
Vedlegg A

Grafer

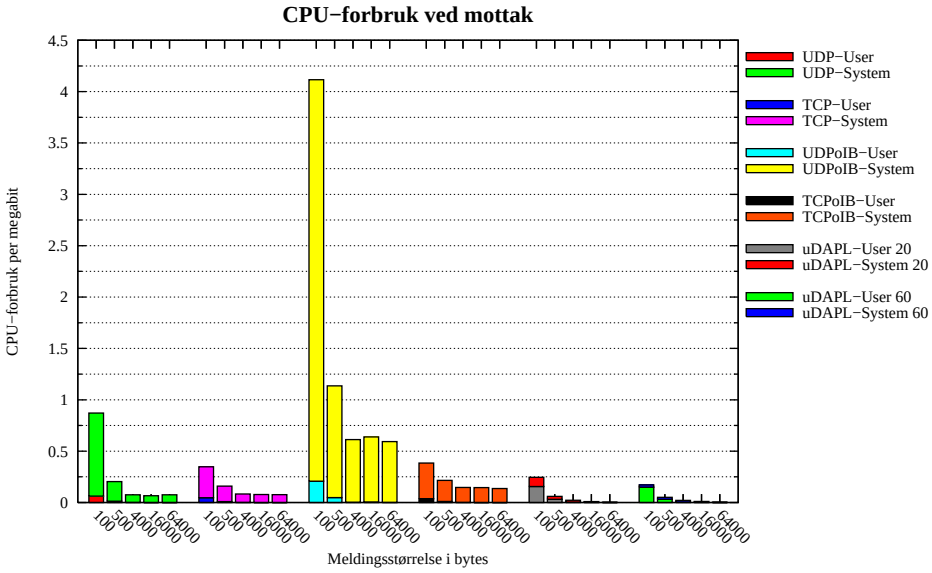
I dette vedlegget vises noen av grafene fra kapittel 9 uten beskjæring.



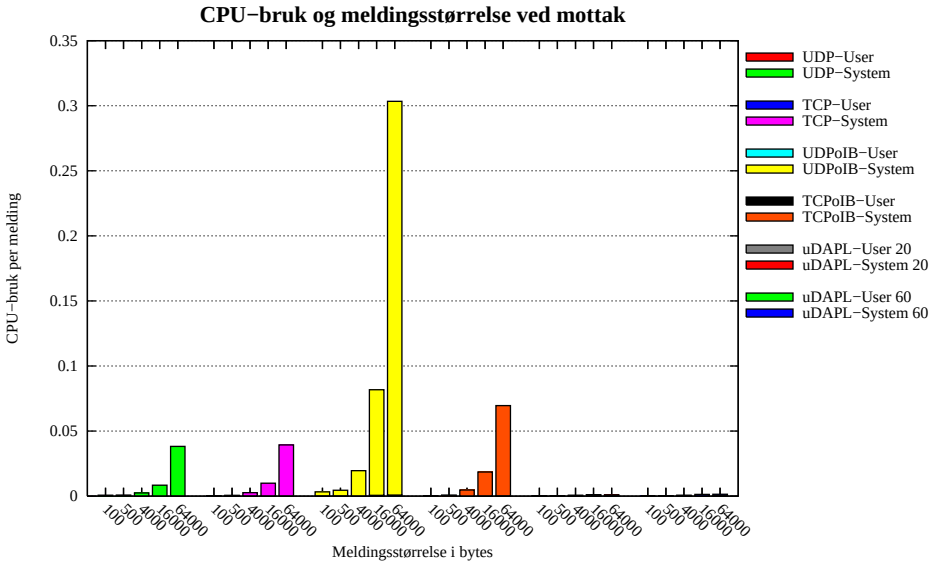
Graf A.1: CPU-forbruk per megabit ved sending av meldinger



Graf A.2: CPU-forbruk per melding ved sending



Graf A.3: CPU-forbruk per megabit ved mottak av meldinger



Graf A.4: CPU-forbruk per melding ved mottak