NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk

Institutt for datateknikk og informasjonsvitenskap

BOKMÅL

Sensurfrist: 13. juni 2012

Løsningsforslag for Eksamen i TDT4190 Distribuerte systemer

Onsdag 23. mai 2012 9.00 – 13.00

Faglig kontakt under eksamen:
Jon Olav Hauglid (735 93 440)

Hjelpemidler:
Hjelpemiddelkode D. Ingen trykte eller håndskrevne hjelpemidler tillatt. Bestemt, enkel kalkulator tillatt.

Oppgavesettet inneholder 10 oppgaver. Det er angitt i prosent hvor mye hver oppgave teller ved sensur. Innenfor hver oppgave teller deloppgavene likt. Les igjennom hele oppgavesettet før du begynner å lage løsning. Disponer tiden godt! Gjør rimelige antagelser der du mener oppgaveteksten er ufullstendig og skriv kort hva du antar. **Lykke til!**

Oppgave 1 – Egenskaper ved distribuerte systemer (10 %)

a) Hva er fordelene ved et distribuert system i forhold til et sentralisert system?

Distribuerte systemer muliggjør deling av ressurser mellom flere brukere. Det gjør det også mulig å slå sammen ressurser som er delt i utgangspunktet eller ressurser som er for mye for en enkelt sentralisert maskin. Gjennom dette kan man oppnå større grad av skalerbarhet og tilgjengelighet. Distribuerte systemer gjør det også mulig med samhandling mellom deltakere (eks: lynmeldinger, videokonferanser).

b) Heterogenitet er en typisk utfordring for distribuerte systemer. Beskriv kort hvordan heterogenitet kan håndteres.

Man kan definere standarder for hvordan (del-)systemer skal oppføre seg og for hvilket grensesnitt de skal ha. Man kan introdusere mellomvare som utligner forskjeller mellom individuelle deltakere. Man kan bruke virtuelle maskiner som gjør det mulig å kjøre samme kode på flere plattformer.

Oppgave 2 – Distribuerte objekter og fjernkall (5 %)

Er det ønskelig å gjøre fjernkall helt transparente (dvs. helt likt lokale funksjons-/metodekall)? Diskuter fordeler og ulemper.

Det forsøkes som regel å gjøre syntaksen så lik som mulig mellom fjernkall og lokale kall, men helt transparent er det i praksis umulig å få til (det tar f.eks. alltid lenger tid å gjøre et fjernkall), og det er som regel heller ikke ønskelig, siden man ønsker utvidet feilhåndtering av fjernkall. I motsetning til lokale kall kan fjernkall feile, f.eks. hvis nettverksforbindelsen går ned eller hvis motparten krasjer. Derfor er det ikke alltid ønskelig å gjøre dem helt transparante, siden slik transparens vil hindre feildeteksjon og -håndtering.

Fordelene ved å gjøre fjernkall transparente, er at man ikke trenger å skille mellom fjernkall og lokale kall og at det derfor blir lettere å skrive og vedlikeholde koden, f.eks. ved at marshalling og unmarshalling gjøres automatisk. Det gjør også grensesnittet mer fleksibelt siden det ikke er nødvendig å endre koden hvis man senere velger å flytte på objekter eller funksjoner. Man slipper også å forholde seg til marshalling og unmarshalling og håndtering av nettverksfeil ved å sende meldinger på nytt.

Ulempene er at man gjemmer bort kompleksitet og semantiske forskjeller mellom fjernkall og lokale kall. Lokale kall har en nøyaktig-en-gang-semantikk, mens fjernkall kan ha flere andre semantikker (maybe, at-least-once, at-most-once). Dette vil ikke komme frem i et helt transparent grensesnitt. Det mye som kan gå galt i et fjernkall, og effektiv håndtering av feilsituasjoner er ikke mulig når kallene er transparente. F.eks. kan det være ønskelig å skille mellom at nettverket går ned og at motparten krasjer.

Gjeldende praksis er at man gjør syntaksen lik, og at forskjellene blir tydeliggjort i grensesnittet, f.eks. ved å deklarere at fjernkall kaster unntak ved feil.

Oppgave 3 – Operativsystem og distribuerte systemer (10 %)

Anta at vi har to klient-tjener-applikasjoner. Typisk bruksmønster for applikasjon A er at en klient kobler seg opp mot tjeneren, sender en forespørsel, får svar og kobler seg ned. For applikasjon B er det derimot mange forespørsler/svar for hver klient-oppkobling. Hvilken trådarkitektur ville du valgt for tjeneren for applikasjon A? Hva med applikasjon B? Begrunn svaret.

A: En tråd per forespørsel og en tråd per forbindelse blir det samme i dette tilfellet. Det gir typisk mye overhead å starte en tråd per forespørsel. Kan derfor være fornuftig med en "worker pool architecture" der forespørsler blir lagt i en felles kø som håndteres av et sett med alltid eksisterende tråder.
B: En tråd per forbindelse gir god mening her ettersom mange forespørsler skal håndteres for en gitt forbindelse. Overhead i forbindelse med starting av tråder bør ikke bli et problem her ettersom samme tråd vil svare på mange forespørsler. Hvis man antar at klienten kan genererere ny forespørsel før svar på forrige, kan det gi mening med flere tråder per forbindelse og benytte samtidige kall ("concurrent invocations").
Gitt fornuftige antagelser utover det som er beskrevet i oppgaveteksten, kan tråd per objekt vurderes for både A og B.

Oppgave 4 – Likemannsnettverk ("Peer-to-peer networks") (10 %)

Oppslag ("routing") i et likemannsnettverk skjer som regel steg for steg. Hvorfor er dette nødvendig? Hvorfor kan ikke oppslag i steden alltid gjøres direkte fra avsender til mottaker? Begrunn svaret.

For at likemannsnettverket skal være skalerbart, kan ikke alle noder vite om alle andre noder. Dette gjelder spesielt ved høy churn. Konsekvensen er at direkte oppslag er umulig og at forespørsler må rutes steg for steg.

Oppgave 5 – Sikkerhet (15 %)

a) Alice og Bob har hver sine offentlige og private nøkler. Av disse fire nøklene, kjenner Alice tre – alle unntatt Bob sin private nøkkel. Gi eksempler på hva Alice kan bruke hver av disse tre nøklene til. (Ett eksempel per nøkkel er nok.)

   AlicePub: Kan gis til Bob slik at Bob kan bruke den til å kryptere meldinger til Alice siden bare AlicePriv kan dekryptere.
   AlicePriv: Kan brukes for å signere meldinger da kun AlicePub vil kunne dekryptere meldingen og dermed bevise at meldingen kom fra Alice.
   BobPub: Kan brukes til å sende krypterte meldinger til Bob da kun BobPriv vil kunne dekryptere meldingen.

b) Tenk at du skal lage en ny krypteringsalgoritme. Ville du holdt algoritmen (og implementasjonen) hemmelig eller ikke? Begrunn svaret.

   Hvis man gjør algoritmen og implementasjonen offentlig, vil flere kunne delta i å finne mulige feil og mangler. Det vil kunne gjøre algoritmen og implementasjonen sterkere og lettere å stole på. Det er nøklene som skal være hemmeligheten – ikke algoritmen.

Oppgave 6 – Distribuerte filsystemer (10 %)

Hvorfor egner Andrew File System (AFS) seg bedre enn Network File System (NFS) til å lagre hjemmekatalogene til ansatte ved et universitet? Vil de samme fordelene gjelde for en student som bruker forskjellige maskiner på forskjellige datasaler? Hvorfor/hvorfor ikke?

En av de største forskjellene mellom AFS og NFS er at AFS har en persistent (dvs. overlever reboot) cache av hele filer på lokal disk, mens NFS bare cacher diskblokker i RAM. En ansatt som nesten utelukkende bruker sin egen kontor-PC vil derfor oppleve at de fleste filene allerede er cachet på lokal disk når han skal bruke dem. Ved bruk må klienten spørre tjenren om det har kommet nye versjoner av filen, og kun i veldig få tilfeller der den ansatte eller andre har endret en fil i hjemmekatalogen fra et annet sted vil det være nødvendig å lese nye data fra tjeneren.

En student som flytter mellom maskiner vil normalt ikke oppleve den samme effekten. Filer blir oppdatert på forskjellige maskiner, så mye av effekten ved å bygge opp en lokal cache med arbeidssettet vil forsvinne. Det betyr ikke at fordlene forsvinner helt, men de blir redusert. Hvis studenten har en fast maskin på hver datasal og det ikke er alt for mange andre brukere som spiser av cachekapasiteten, vil det imidlertid være mulig å bygge opp et rimelig arbeidssett på hver av dem og bare oppdatere endringene når det blir nødvendig. Det kan også være en fordel å bruke AFS selv om cachen må bygges opp for hver sesjon, f.eks. hvis studenten gjør operasjoner som bruker filene mye. Filene vil da ligge lokalt i stedet for å hentes over nett (hvis blokkcachen i NFS ikke er stor nok til å holde alt).

Oppgave 7 – Tid og global tilstand (10 %)

Anta et distribuert system med to prosesser p1 og p2. Prosess p1 har en lokal historie med fire hendelser e1-e4 og p2 har en lokal historie med tre hendelser f1-f3. Prosess p1 har en lokal variabel v1 og p2 har en lokal variabel v2. Begge variablene har verdien 0 i utgangspunktet. Detaljer om hendelsene er gitt under:

Hendelse	**Vektorklokkeverdi**	**Verdi av v1**
e1	(1,0)	4
e2	(2,0)	5
e3	(3,0)	5
e4	(4,3)	4
Hendelse	**Vektorklokkeverdi**	**Verdi av v2**
f1	(1,1)	5
f2	(2,2)	4
f3	(2,3)	4

Har summen av v1 og v2 noen gang vært 10? Begrunn svaret.

Bruker Sxy for global tilstand der sist skjedde hendelse i p1 er ex og sist skjedde hendelse i p2 er fy. Vi får disse mulige tilstandsovergangene:
S00 => S10
S10 => S20 eller S10 => S11
S20 => S30 eller S20 => S21
S11 => S21
S30 => S31
S21 => S31 eller S22
Osv.
I både S21 og S31 er summen lik 10. Siden vi ikke kan unngå å gå innom enten S21 eller S31, vet vi "definitly" at summen har vært 10 (ikke bare "possibly").

Oppgave 8 – Koordinering og enighet (10 %)

Forklar ved hjelp av et eksempel hvordan multicast-meldinger kan være FIFO-ordnet uten å samtidig være kausalt ordnet.

FIFO ordner bare meldinger sendt fra samme avsender. Anta at node A sender to meldinger. Disse to meldingene blir levert i samme rekkefølge hos nodene A, B og C. Vi har dermed FIFO-ordning. Men anta at C sender en melding etter å ha fått levert A sin første melding (denne meldingen er dermed kausalt avhengig av As første melding). Anta at denne meldingen blir levert hos node B før As første melding. Vi har da ikke kausal ordning (men fremdeles FIFO-ordning).

Oppgave 9 – Distribuerte transaksjoner (10 %)

Hvilket problem bruker man 2PC for å løse? Hvorfor må deltakerene logge (til disk) beslutninger som tas før meldinger sendes?

To-fase-commit brukes for å koordinere deltakere i distribuerte transaksjoner. Det sikrer at alle deltakere er enige om commit/abort og at en transaksjon ikke introduserer inkonsistens ved at den kun har effekt hos noen av deltakerene.
Logging til disk før meldingssendelse gjøres for å forhindre at deltakere kan ombestemme seg etter krasj. Hvis melding sendes før logging og en node krasjer etter at den har sendt en melding men før den har logget meldingen, kan den ombestemme seg når den restarter fordi den ikke har noen anelse om beslutningen den tok (og informerte andre om) før den krasjet.

Oppgave 10 – Google case study (10 %)

Både Google File System (GFS) og Bigtable bruker sentraliserte mastere, men skalerer likevel godt. Hvorfor fungerer dette? Hvilke fordeler er det ved å gjøre systemene avhengige av en sentralisert master?

Det fungerer fordi trafikken mellom klient og master er redusert til et minimum, slik at det aller meste av trafikken kan gjøres direkte mellom klient og slaver uten å involvere masteren. I GFS og Bigtable gjøres dette ved at masterene bare håndterer metadata og administrative oppgaver, mens all lesing og skriving av data går direkte mot slavene.

Fordlene med sentralisert master er at det reduserer behovet for koordinering og distribuert enighet. Når det bare er én master, er denne den autorative kilden for alle, og den trenger ikke bli enig med andre for å kunne svare på forespørsler. Dette gjør f.eks. at masteren har overblikk og kan gjøre lastbalansering både av CPU-last og datalagring.

Sentralisert master gir også en mye enklere arkitektur, noe som øker utviklernes og brukernes forståelse og reduserer mulighetene for feil. Klare skiller i arkitekturen gjør også systemet mer fleksibelt og utvidbart. F.eks. har Google pga. stor last utvidet til multi-master uten å endre den overordnede arkitekturen.