NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk

Institutt for datateknikk og informasjonsvitenskap

Løsning på eksamen i TDT4190 Distribuerte systemer
Fredag 28. mai 2004, 0900-1300

Det ønskes korte og konsise svar på hver av oppgavene. Det vesentlige er å kunne dokumentere forståelse, beherske prinsipper og se sammenhenger - ikke å kunne gjengi en mengde detaljer.

Der det synes å mangle noen opplysninger, må det angis hvilke antagelser som synes å være naturlige. Merk at viktige begreper er angitt på både norsk og engelsk.

Oppgave 1 – Modeller og standarder (Models and standards) – 25%

a) Beskriv kort klient-tjener begrepet (client server)

SVAR:

Plasser ulike programmoduler – presentasjon/beregning/aksessering – på ulike noder:

- Tre-nivås => P (K) + B (TK) + A (T)
- To-nivås, fet tjener => P (K) + B & A (T)
- To-nivås, fet klient => P & B (K) + A (T)
- Ett-nivås => P & B & A (TK)

Videre kan det eventuelt utnyttes muligheter for:

- Parallellisering – F.eks. presentasjon
- Fragmentering – F.eks. beregning
- Replisering – F.eks. aksessering

b) Diskuter kort hvorfor objektorientering (object orientation) egner seg til implementasjon av klient-tjener modellen spesielt

SVAR:

Sammenkoplingen av prosedyrer og data, innkapslingen som skillet mellom grensesnitt og implementasjon medfører, samt invokeringen av metoder ved å sende meldinger til objekter

c) Angi kort hvordan mellomvare arkitekturer (middleware architectures) brukes til å implementere distribuerte systemer generelt

SVAR:

En konstruerer et programvare lag (med et generelt API) mellom globale applikasjoner og lokale ressurser slik at en gitt applikasjon kan nå en vilkårlig lokal ressurs, og en gitt ressurs kan nås av en vilkårlig applikasjon – hvor mellomvare laget står for ruting av forespørslene til riktig node og oversetting av forespørslene til riktig språk

d) Sammenlign kort minst tre ulike mellomvare standarder

SVAR:

Her forventes det altså ikke mer enn tre av for eksempel de syv nedenforstående.

ISO/RM-ODP

* OO-type, rammeverk-nivå
+ Omfatter det meste som trengs
- Enormt begrepsmessig apparat

OMG/CORBA

* ORB-type, arkitektur-nivå
+ OO-basert
- Mye å sette seg inn i

OSF/DCE

* RPC-type, arkitektur-nivå
+ Enkel å ta i bruk
- Ikke OO-basert

MS – DCOM/OLE

* RPC/OO-mellomting, blanding av alt
+ MS støtter det
- MS bestemmer alt

X/OPEN – TX/XA

* DTP-fokus, blankt på mye
+ Tidlig fokus på transaksjoner, således litt oppdragende på andre
- Inkludert i andre etter hvert, således litt overflødig i seg selv

TINA

* Telekom-rettet, spesialisering av RM-ODP
+ Effektivt verktøy i sin valgte nisje
- RM-ODP++, for mye av det gode !

MSS

* Multimedia-rettet, spesialisering av CORBA
+ Effektivt verktøy i sin valgte nisje
- CORBA++, for mye av det gode ?

Oppgave 2 – Distribuerte databasesystemer (Distributed database systems) – 25%

a) Beskriv kort transaksjonsbegrepet (transactions)

SVAR:

Transaksjoner er programvare enheter som skal fungere som

- Arbeidsenheter – dvs. at alle leste verdier forblir lokalt brukbare underveis
- Samtidighetsenheter – dvs. at alle skrevne verdier forblir globalt synlige først etterpå
- Rekonstruksjonsenheter – dvs. at alle skrevne verdier forblir angrbare / bekreftbare

med henblikk på ekstern parallellitet og / eller interne feil. Problemer som transaksjoner da håndterer er:

- Manglende atomiskhet – dvs. at transaksjoner ikke er udelelige
- Manglende konsistens – dvs. at transaksjoner ikke bevarer konsistens
- Manglende integritet – dvs. at transaksjoner ikke er isolerbare
- Manglende varighet – dvs. at transaksjoner ikke overlever sammenbrudd

b) Sammenlign kort en nøstet distribuert modell (nested distributed model) med en flat distribuert modell (flat distributed model) for distribuerte databasesystemer

SVAR:

I en flat distribuert modell håndteres alle operasjoner knyttet til en gitt node av en enkelt prosess på den noden, mens i en nøstet distribuert modell kan operasjonene knyttet til en gitt node fordeles på en vilkårlig (trebasert) struktur av separate delprosesser

c) Diskuter kort ulike måter å oppnå atomiskhet (atomicity) på i en flat distribuert modell

SVAR:

Noen viktige måter:

2PW:

- Alle objekter som skal oppdateres på en node må skrives til en logg på noden før noe objekt blir skrevet til nodens database

  Håndterer feil innen en node på en effektiv måte, håndterer ikke feil mellom noder

2PC:

- Utnytter en todelt avslutning med en 1. fase m/stemming og en 2. fase m/utføring etter alle-har-veto prinsippet koplet til enten feil eller autonomi

  Sikrer autonomi og effektivitet i normaltilfeller uten mye feil, mindre overhead

3PC:

- *3PC* legger på en ekstra fase i.f.t. *2PC* hvor nodene blir forberedt på det endelige resultatet gitt at ingen feil oppstår

  Sikrer autonomi og effektivitet i unntakstilfeller med mye feil, mer overhead

d) Angi kort hvilke endringer som trengs for å oppnå atomiskhet i en nøstet distribuert modell

SVAR:

2PW blir som før, og 3PC fungerer relativt til 2PC som før.

2PC - Det foregår en flat 2PC mellom tretoppene. Hver av tretoppene gjør følgende:

- *Før oppgaveutføring*:
  Fordeler oppgaver på sine barn som igjen fordeler deloppgaver på sine barn igjen osv.

- *Under oppgaveutføring*:
  Tentative avgjørelser tas delvis avhengig / dels uavhengig av barns avgjørelser
  (Ett / flere barn: Tentativ ja => Forelder tentativ ja/nei
  Ett / flere barn: Tentativ nei => Forelder tentativ ja/nei
  (Forelder: Tentativ ja => Hvert barn som før
  Forelder: Tentativ nei => Alle barn effektivt abortert)

- *Før avslutningsstemming med ja*:
  Sjekk om alle med tentative ja fortsatt OK
  (Utelatt: Barn av foreldre med tentative nei)

- *Under avslutningsutføring med bekreft / abort*:
  Videresend bekreft /abort til alle med tentative ja
  (Utelatt: Alle allerede abortert direkte / indirekte)

Oppgave 3 – Distribuert pålitelighet (Distributed reliability) – 25%

a) Beskriv kort begrepet distribuert pålitelighet

SVAR:

Håndtering av feil i et distribuert system

Ulike modeller:
- Nærmest forhindre – Ubegrenset redundans !
- Faktisk tolerere – Maskering i ulike former, med gradering av ytelse og funksjonalitet !
- Endog opprette – Begrenset redundans !

(Med faktisk toleranse som det naturlige valg i en eller annen form)

b) Diskuter kort de ulike utfordringene som må håndteres for å oppnå distribuert enighet (distributed agreement)

SVAR:

Ulike dimensjoner:
- Ikke bysantinsk / Bysantinsk – Kan meldingsinnhold stoles på ?
- Signert / Usignert – Kan meldingsavsender spores ?
- Synkront / Asynkront – Kan meldingstid begrenses ?

(Med synkron antagelse for asynkron realitet)

Ulike resultater:
- Synkront, ikke bysantinsk => $N \geq M+1$
- Synkront, bysantinsk, signert => $N \geq 2M+1$
- Synkront, bysantinsk, usignert => $N \geq 3M+1$
- Asynkront => Umulig

(M.h.t. krav til antall noder N vs. antall feilende noder M)

c) Angi kort hvordan bruk av redundans (redundancy) kan utnyttes for å oppnå distribuert pålitelighet

SVAR:

En logisk ressurs implementeres som flere fysiske ressurser
Gruppeaktig replisering reduserer avhengigheten av en enkelt fysisk ressurs

HW / SW: Med aktiv eller passiv redundans
SW: Så vel data som applikasjon

Kan gi mer feiltoleranse, mer tilgjengelighet og mer ytelse (HW / SW)
Kan gi problem m.h.t. full transparens og/eller full konsistens (SW)

d) Sammenlign kort de ulike problemene som må løses for å oppnå distribuert pålitelighet henholdsvis med og uten kommunikasjonsfeil (communication failures)

SVAR:

Uten replisering – Uten kommunikasjonsfeil:
- Ingen store utfordringer

Uten replisering – Med kommunikasjonsfeil:
- 3PC-terminering av transaksjoner kan gi blokkering i forsøk på å oppnå atomiskhet

Med replisering – Uten kommunikasjonsfeil:
- 2PL-låsing for transaksjoner må få utvidelser for å sikre kollisjon av tilhørende låser

Med replisering – Med kommunikasjonsfeil:
- MajKonsensus- / VirtPartisjon-algoritmer må til for å utpeke partisjon for oppdatering
- 2PL-låsing for transaksjoner må få utvidelser for å sikre kollisjon av tilhørende låser
- 3PC-terminering av transaksjoner kan gi blokkering i forsøk på å oppnå atomiskhet

Oppgave 4 – Distribuert delt lager (Distributed shared memory) – 25%

a) Beskriv kort begrepet distribuert delt lager

SVAR:

Simulering av felles lager i en omgivelse uten felles lager

En multiprosessor – med fysisk delt lager – er lett å programmere (SW-messig)
En multimaskin – uten fysisk delt lager – er lett å konstruere (HW-messig)

Distribuert delt lager – simulert, logisk delt lager – søker det beste fra begge verdener

b) Sammenlign kort implementasjon av distribuert delt lager i henholdsvis maskinvare (hardware), operativsystem (operating system) og kjøretidssystem (runtime system)

SVAR:

Maskinvare:
- Blokkbaserte overførsler
- Fjernaksess og caching i maskinvare

Operativsystem:

- Sidebaserte overførsler
- Fjernaksess og caching i programvare

Kjøretidssystem:

- Postbaserte overførsler
- Fjernaksess og caching i programvare

Mellomklasse: Dash (som eksempel på NUMA-CC) regnes som OS-implementasjon med sidebaserte overførsler, med caching i programvare, men med fjernaksess i maskinvare

c) Angi kort ulike måter å oppnå konsistens (consistency) på i distribuert delt lager

SVAR:

Hovedsakelig to ulike måter:

- Gradsavballansering – dvs. kopling av konsistens til anvendelser ved å avgjøre i hvilken grad DSM må være konsistent

  Prosessor konsistens:
  Alle prosessorer ser lokale operasjoner i samme rekkefølge

  Kausal konsistens:
  Alle prosessorer ser lokale og koplete, globale operasjoner i samme rekkefølge

  Sekvensiell konsistens:
  Alle prosessorer ser lokale og globale operasjoner i samme rekkefølge

  Altså gradvis tyngre krav: PK => KK => SK

- Tidsavballansering – dvs. kopling av konsistens til synkronisering ved avgjøre til hvilke tider DSM må være konsistent

  Båndleggingskonsistens:
  Fellesdata gjøres konsistente før kritiske regioner

  Frigjøringskonsistens:
  Fellesdata gjøres konsistente etter kritiske regioner

  Svak konsistens:
  Fellesdata gjøres konsistente både før og etter kritiske regioner

  Altså ikke helt gradvis tyngre krav: BK => SK & FK => SK

d) Diskuter kort hvordan distribuert delt lager implementeres i IVY/MIRAGE

SVAR:

Sidebasert (dvs. OS-simulering) – Sekvensiell konsistens (som lagringskrav)

Globale sider er spredt på lokale noder – Ingen eier en gitt side permanent
En gitt side kan være skitten (W-tilstand) eller ren (R-tilstand)
En skitten side vil finnes i en kopi – En ren side kan finnes i flere kopier
En ønsket side kan være på tilsvarende node eller ikke

Resultat: 6 ulike tilfeller for leseaksess & 6 ulike tilfeller for skriveaksess
Eier-lokalisering: Benytt sentral forvalter / Forfølg sannsynlig eier
Kopi-lokalisering: Forvalter evt. eier(e) holder kopiliste(r) / Forespørrer benytter kringkast.
Flere aktive aktører gjør spontane invalideringer mulig: Behov for nye sideutbyttingsalg.